6-1-2008

A homography-based multiple-camera person-tracking algorithm

Matthew Robert Turk

A Homography-Based Multiple-Camera Person-Tracking Algorithm

by

Matthew Robert Turk

B.Eng. (Mech.) Royal Military College of Canada, 2002

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in the Chester F. Carlson Center for Imaging Science
Rochester Institute of Technology

12 June 2008

Signature of the Author _____

Accepted by _____
Coordinator, M.S. Degree Program Date

CHESTER F. CARLSON CENTER FOR IMAGING SCIENCE
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK, UNITED STATES OF AMERICA

CERTIFICATE OF APPROVAL

M.S. DEGREE THESIS

The M.S. Degree Thesis of Matthew Robert Turk
has been examined and approved by the
thesis committee as satisfactory for the
thesis required for the
M.S. degree in Imaging Science

Dr. Eli Saber, Thesis Advisor

Dr. Harvey Rhody

Dr. Sohail Dianat

Date

THESIS RELEASE PERMISSION
ROCHESTER INSTITUTE OF TECHNOLOGY
CHESTER F. CARLSON CENTER FOR IMAGING SCIENCE

Title of Thesis:
A Homography-Based Multiple-Camera Person-Tracking Algorithm

I, Matthew Robert Turk, hereby grant permission to the Wallace Memorial Library of RIT to reproduce my thesis in whole or in part. Any reproduction shall not be for commercial use or profit.

Signature _____
Date

A Homography-Based Multiple-Camera Person-Tracking Algorithm

by

Matthew Robert Turk

Submitted to the
Chester F. Carlson Center for Imaging Science
in partial fulfillment of the requirements
for the Master of Science Degree
at the Rochester Institute of Technology

Abstract

It is easy to install multiple inexpensive video surveillance cameras around an area. However, multiple-camera tracking is still a developing field. Surveillance products that can be produced with multiple video cameras include camera cueing, wide-area traffic analysis, tracking in the presence of occlusions, and tracking with in-scene entrances.

All of these products require solving the consistent labelling problem. This means giving the same meta-target tracking label to all projections of a real-world target in the various cameras.

This thesis covers the implementation and testing of a multiple-camera people-tracking algorithm. First, a shape-matching single-camera tracking algorithm was partially re-implemented so that it worked on test videos. The outputs of the single-camera trackers are the inputs of the multiple-camera tracker. The algorithm finds the feet feature of each target: a pixel corresponding to a point on a ground plane directly below the target. Field of view lines are found and used to create initial meta-target associations. Meta-targets then drop a series of markers as they move, and from these a homography is calculated. The homography-based tracker then refines the list of meta-targets and creates new meta-targets as required.

Testing shows that the algorithm solves the consistent labelling problem and requires few edge events as part of the learning process. The homography-based matcher was shown to completely overcome partial and full target occlusions in one of a pair of cameras.

Acknowledgements

- The Canadian Air Force made this work possible through the Sponsored Post-Graduate Training Program.
- Professor Warren Carithers suggested the use of a function used in the Generator program, which was used for testing the algorithm.
- Mr. Sreenath Rao Vantaram supervised the segmentation of all real-world video sequences.
- Finally, Ms. Jacqueline Speir helped me to clarify and expand many of the concepts discussed herein.

Contents

Chapter 1

Introduction

Video surveillance is a difficult task. Based on the field of computer vision, itself only a few decades old, the automatic processing of video feeds often requires specialized encoding and decoding hardware, fast digital signal processors, and large amounts of storage media.

The need to process multiple video streams is becoming more important. Video camera prices continue to drop, with decent "webcams" available for less than twenty dollars. Installation is similarly inexpensive and easy. Furthermore, social factors are assisting the spread of surveillance cameras. City police forces, such as those in London and Boston, and private businesses, such as shopping malls and airports, are using recent terrorism to justify increasing video surveillance. In most major cities it is now easy to spot video cameras. Some installations even boast

low-light capabilities using cameras sensitive to near- or thermal-infrared wavelengths.

Despite the increasing prevalence of multiple camera surveillance installations, few algorithms extract additional, meaningful multiple-camera tracking information. Chapter 2 will cover a few of the algorithms that track moving objects in a single video stream. Solutions to the single-camera tracking problem are fairly well developed. However, multiple-camera surveillance systems demand algorithms that can process multiple video streams.

1.1 Motivating example

As a motivating example, consider the overhead view of a surveilled area as seen in Figure 1.1. Cameras A and B are disjoint – they look at different areas of the world and do not overlap. However, cameras A and C partially overlap, as do cameras B and C. An object in either of the darker overlapping areas will be visible to two cameras simultaneously.

Now examine the output of the three cameras. There are two people in the world. However, between the three cameras they have been given four different labels: A-8, B-2, C-4, and C-5. Given these object labels, the most important piece of information that we could find is which labels refer to the same real-world objects. This is the consistent labelling problem.

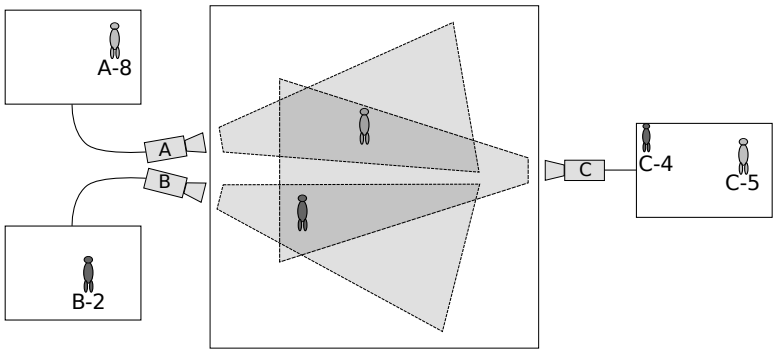
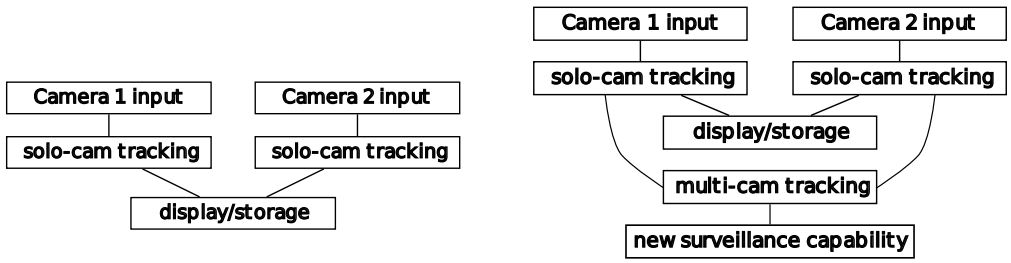


Figure 1.1: Three cameras look at the same general area in this overhead view. Across the three cameras, two targets are given four tracking labels.

Humans are fairly good at solving the consistent labelling problem, up to a certain point. Human surveillance operators can keep a mental model of the locations of the cameras in the world, and can often match features from camera to camera even if different camera modalities are used (e.g. one RGB camera and one thermal infrared camera). Additionally, humans are much better than computers at matching objects, even if the objects are observed from widely disparate viewpoints and consequently have different appearances. However, using humans to analyse multiple video streams does not scale well since a person can only look at one screen at at time, even though there may be many relevant views of a scene. If multiple surveillance operators are used, each one responsible for a particular area, then the system would require the development of procedures for control, target tracking, target handoff, and possible operator inattention.



(a) Recording the output of single-camera trackers is common, but does not take advantage of having multiple cameras.

(b) Multiple cameras can give additional information and increase surveillance potential.

Figure 1.2: Algorithms that effectively use multiple cameras can produce extra useful information.

An important task for a surveillance system is being able to track a target of interest as it moves through a surveilled area. Many cameras might have the target in view at any given time, but the conscious effort required of a human to determine this set of cameras is not trivial, even if only a handful of cameras are used. Additionally, the set of cameras viewing the target changes constantly as the target moves. If the consistent labelling problem is solved, and the computer knows whether a target should appear in each camera's field of view, then the computer can automatically cue the correct set of cameras that show the target.

Figure 1.2 illustrates the difference between algorithms that treat multiple cameras as a group of single cameras, and those that treat the cameras as something more. The algorithms that fall into Figure 1.2(a) do not

care that multiple cameras might view the same part of the world. The second class of algorithms, shown in Figure 1.2(b), take the outputs of single camera trackers and combine them. New surveillance capabilities are created. Some examples of the capabilities created by these multi-camera-aware algorithms are mentioned below.

1.2 Scope – goals

This thesis covers the development, implementation, and testing of a multiple camera surveillance algorithm. The algorithm shall have the following characteristics:

1. Independent of camera extrinsic parameters, *i.e.* location and orientation. The algorithm should smoothly handle widely disparate views of the world.

2. Independent of camera intrinsic parameters, *i.e.* focal length, pixel skew, and location of principal point. Different cameras are available on the market – the algorithm should be able to handle multiple focal lengths, differences in resolution, and the like.

3. Independent of camera modality. The algorithm should be able to handle the output of any single-camera tracker. The algorithm should not depend on whether the underlying camera hardware is

RGB, near-infrared, thermal infrared, or some other image-forming technology.

4. Solve the consistent labelling problem. One real-world target should be linked to one object label in each of the cameras in which that target is visible.

5. Robust to target occlusions and in-scene entrances. If a target enters the surveilled area in the middle of a scene, say, through a door, then the algorithm should correctly solve the consistent labelling problem. Similarly, if one target splits into two, as when two close people take different paths, the algorithm should identify and correctly label the two targets.

6. Simple to set up. No camera calibration should be required. Training, if needed, should take as little time as possible and should be done with normal in-scene traffic. Training should be automatic and should not require operator intervention.

7. Capable of camera cueing. The algorithm should be able to determine which cameras should be able to see a given target.

1.3 Scope – limitations

The scope of the algorithm shall be limited as follows:

1. The algorithm shall be used for tracking walking people. Vehicles, animals, and other classes of moving objects are not included in the scope of this thesis.

2. Pairs of cameras to be processed will have at least partially overlapping fields of view. This requires the operator to make an initial judgement when installing the hardware and initializing the algorithm: to decide which cameras see the same parts of the world.

3. The cameras shall be static. Once installed, both the intrinsic and extrinsic parameters of the camera shall be fixed. This means that a camera can not be mounted on a pan-tilt turret, or if it is, the turret must not move.

4. The output images of the video cameras will be of a practical size. The algorithm will not include single-pixel detectors (*e.g.* infrared motion detectors, beam-breaking light detectors). This limitation is necessary to ensure that single-camera tracking is possible without major changes to the chosen algorithm.

5. Frame rates will be sufficient to allow the single-camera tracking algorithm to work properly.

6. The cameras shall approximate regular central-projection cameras with basic pinhole optics. Cameras with extremely wide fields of

view – fisheye lenses – or significant un-corrected distortions will not be used.

7. Most importantly, the targets shall be walking on a ground plane. The overlapping area between any two cameras shall not have significant deviations from a planar surface. Code to deal with hilly areas or steps shall not be included in this algorithm.

8. The cameras shall not be located on the ground plane. This prevents a degenerate condition in the scene geometry, as shall be shown later.

1.4 Contributions to field

As mentioned above, multiple-camera video processing is a relatively new domain. Algorithms are constantly under development, and there are many problems yet to solve. If the algorithm developed in this document satisfies the goals and limitations described in Sections 1.2 and 1.3, then the following scenarios will be made possible:

- Automatic cueing: A target of interest walks into a surveilled area. The operator marks the target in one camera. As the target moves throughout the area, the computer, driven by the algorithm, automatically shows all the video feeds in which the target is visible.

The target could be marked by a consistently-coloured "halo" or a bounding box. This lets the operator concentrate on the target's actions, rather than on its position in the world relative to every camera.

- Path analysis: An area is placed under surveillance. Rather than trying to manually match the paths of people from camera to camera, the algorithm automatically links the paths taken by the people moving through the area. This enables flow analysis to be carried out quicker and more effectively.

- Tracking with occlusion recovery. To fool many current tracking algorithms, move behind an occlusion (*e.g.* a building support pillar or a tall accomplice), change your speed, and then move out of the occlusion. Occlusion breaks many current tracking algorithms, and most others break if the speed change is significant. So long as the target remains visible in at least one camera, the algorithm discussed in the forthcoming chapters shall recover from occlusions and shall re-establish the consistent tracking label.

- In-scene entrances. The algorithm shall be able to create consistent tracking labels in scenes where people can enter in the middle of a frame, such as through an elevator or a door.

1.4.1 Specific contributions

This thesis provides these specific contributions to the field of video processing:

- A method to find the feet feature of a target even when the camera is significantly tilted,
- A method to use target motion to find a plane-induced homography even when entrances and exits are spatially limited, and
- A method with specific rules describing how to use a plane-induced homography to create and maintain target associations across multiple cameras.

The underlying theory is discussed in Chapter 3, with implementation details in Chapter 4. Test results are shown in Chapter 5.

Chapter 2

Background

2.1 Single camera tracking

Multiple-camera tracking can be done in a variety of ways, but most methods rely on the single camera tracking problem being previously solved. Indeed, except for those multiple-camera tracking papers that aim to simultaneously solve the single- and multiple-camera tracking problems, most papers include a statement to the effect of "we assume that the single-camera tracking problem has been solved." However, the present work requires the problem to be *actually* solved, not just *assumed* solved. This is because a working system is to be implemented for this thesis, not just conceived.

A recent survey of single-camera tracking algorithms is found in [1].

Plainly, a significant amount of thought has been applied to tracking objects in a single camera's video stream. When successful, the output is a series of frames with regions identified by unique identifiers. Two image regions in frames t and $t+1$ will have the same identifier i if the algorithm has determined that the regions are the projections of the same real-world target object. The single-camera tracking problem is how to determine which regions in frame $t+1$ to mark with which identifiers. The survey shall not be replicated here. Instead, we describe a few characteristic approaches to single camera tracking.

One main type of single-camera tracker calculates a feature vector based on the target, and looks for regions in the next frame that produce a similar feature vector. The idea of kernel-based object tracking is introduced in [2]. In this algorithm a feature vector is calculated from the colour probability density function of an elliptical target region. It is also possible to use a feature vector based on the target's texture information, edge map, or some combination of those and other features, although those were not tested. In the next frame a number of PDFs are calculated for candidate regions around the target's original location. The candidate regions might have a different scale from the original target, to take into account looming or shrinking targets. A distance metric is calculated between the original PDF and each of the candidate PDFs. In this case the metric is based on the Bhattacharyya coefficient. The candidate

target with the smallest distance to the original target is declared to be the projection of the same real-world target, and the appropriate identifier is assigned. It is noted in [2] that other prediction elements can be incorporated into the algorithm, such as a Kalman filter to better predict the target's new location. Using other prediction elements means that fewer candidate PDFs must be calculated.

Another type of tracker looks for regions in frame $t+1$ that have a similar shape to the target in frame t. A triangular mesh is laid over a target in [3]. The texture of the object (*i.e.* its colour appearance) is then warped based on deformations of the underlying mesh. The warped texture is then compared to regions in frame $t+1$ with a metric that incorporates both appearance and mesh deformation energy, and the closest match is called the same object.

Moving farther into shape matching methods, [4] discusses an active shape model for object tracking. Their algorithm uses a principal components analysis to find important points on the boundary of an object in either grayscale or a multi-colour space. The points are then transformed using an affine projection into the coordinate space of frame $t+1$, and compared with the local textural information. If the transformed points are close to edges then the model is valid. If points are projected far from edges then the model is less likely to be correct. Eventually the best model is selected, yielding the new shape of the object in frame $t+1$.

Another method of shape matching, used in this thesis, is introduced in [5]. The method first uses background subtraction to identify foreground blobs. For basic motion the algorithm assumes that the projections of each target do not move very far from frame to frame, so they contain overlapping points. If a target blob in frame t and a candidate blob in frame $t+1$ overlap each other but not other blobs, then they are identified as the same target. If two target blobs merge into one blob in frame $t+1$ due to a partial occlusion then the algorithm uses a shape matching technique introduced in [6]. B-splines are fitted to each original target blob in frame t and to groups of regions in frame $t+1$. The regions are found using a segmentation algorithm shown in [7]. The distances between the B-spline control points of the target blob and each set of candidate regions are compared, then the closest-matching regions are given the same target identifier.

2.2 Multiple camera tracking

When dealing with more than one camera, it is possible that any given pair of cameras may observe the same area or different areas in the world. The latter case is discussed first.

2.2.1 Disjoint cameras

Algorithms that attempt to associate targets across sets of disjoint cameras use a variety of methods. Some, such as [8], use the entry and exit times from each camera to learn the structure of the camera network. From this structure they are able to determine which pairs of cameras view the same regions of the world, and which are disjoint. The system can match targets between disjoint cameras so long as people take a consistent amount of time to pass between cameras. A similar approach to pass target tracks between cameras is used in [9], even if the targets are invisible some of the time (*e.g.* between cameras or occluded).

In [10], a set of randomly sampled target point correspondences is used to create a homography between pairs of cameras. As training data is accumulated they are able to improve the homography in cases where the cameras actually overlap, and discard relationships between disjoint cameras. Their system is thus capable of projecting the location of a target in one camera to its location in another camera, assuming that both cameras observe the target.

Three papers by Javed *et al* take a somewhat more appearance-based approach to disjoint cameras [11, 12, 13]. The algorithm finds a brightness transfer function that projects a target feature vector from one camera to another. Simultaneously, a Parzen window technique is used to estimate the spatio-temporal relationship between entry and exit events in

the various cameras. The projected appearance of a target is then matched from camera to camera using a Bhattacharyya-distance based metric, and the results combined with the spatio-temporal matcher to create target matches.

The present research does not deal with pairs of disjoint cameras. Rather, we assume that the system has been set to only find matches between cameras that have overlapping fields of view. If this level of installation knowledge is unavailable, one of the aforementioned techniques could be used to determine which cameras are disjoint and which overlap.

2.2.2 Pure feature matching

Multi-camera feature matching algorithms assume that some features of a target will remain invariant from camera to camera. A very basic colour matching method is used to identify projections of the same real-world target in different cameras [14]. A matching algorithm on the colour histogram taken along the vertical axis of an object is used in [15].

The method of [16] attempts to reconstruct 3D scene information by creating associations between posed surface elements (surfels) using texture matching. This method has not seen much attention because of the relative complexity of computing scene flow – the 3D equivalent of optical flow.

2.2.3 Calibrated and stereo cameras

If calibration data is available then it is possible to calculate target features and use those features to match targets between cameras, even if they are disjoint. Depending on the algorithm, calibration usually means finding the camera's internal and external parameters (focal length, field of view, location, pose, etc.). For instance, [17] uses the height of targets as a feature that will not change significantly as a target moves from one camera to another. Both the target's height and the colours of selected parts of a target are used to match objects between cameras [18].

The principal axis of a target is the main feature used to match between cameras [19]. The principal axis is transformed from camera to camera using a homography found by manually identifying corresponding ground point pairs in both cameras.

In [20], a system is described that is used to track people and vehicles conducting servicing tasks on aircraft. The system combines appearance-based methods and the epipolar geometry of calibrated cameras to find the 3D locations of targets in the scene, and match them accordingly. Figure 2.1 shows the epipolar geometry relation between two views. Essentially, a 2D point in one camera can be projected to a one-dimensional line in the other camera, upon which the image of world-point $\mathbf{X}$ will lie.

Rather than using epipolar geometry, [21] uses sets of stereo cameras. Each pair of stereo cameras generates depth maps that can be combined

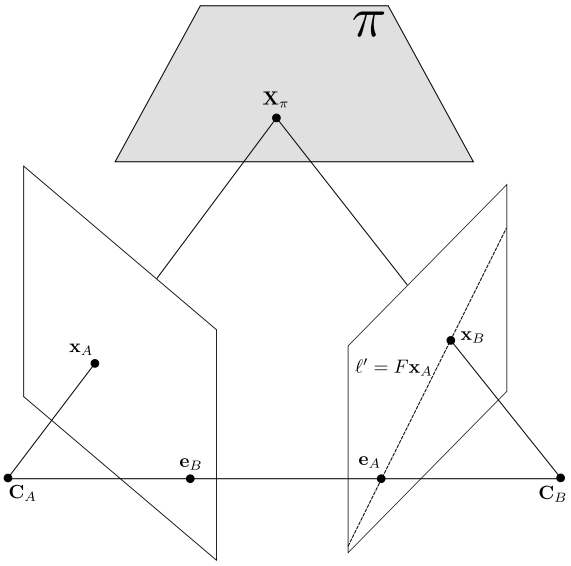


Figure 2.1: Epipolar geometry: the fundamental matrix F links the image points of 3D world point $\mathbf{X}$ in two cameras by $\mathbf{x}_B F \mathbf{x}_A = 0$. $\mathbf{e}_A$ and $\mathbf{e}_B$ are the epipoles – the images of the other camera's centre.

with appearance-based matching to create maps of likely target locations. The map for each camera can be transformed until the cameras line up, at which point full multi-camera tracking can be conducted.

Two papers, [22] and [23], describe methods that require the operator to specify corresponding ground point pairs in cameras. Those point pairs lead to a homography between the cameras that can be used to match targets based on their projected locations in each camera, although no rules are given on how this is done.

2.2.4 Un-calibrated overlapping cameras

One of the more important recent papers in multi-camera tracking is [24], by Khan and Shah. This thesis builds on concepts and methods from that paper. Essentially, the algorithm, discussed at length in Chapter 3, records entry and exit events in one camera, eventually finding the lines created by projecting the edges of the camera's field of view onto a ground plane. Figure 2.2 illustrates this geometry. Once those field of view (FOV) lines are known, target matches can be created between cameras whenever an object crosses one of those lines. An improvement to the original work is made in [25] by re-defining when an entry or exit event is detected – this improvement is discussed in Chapter 3.

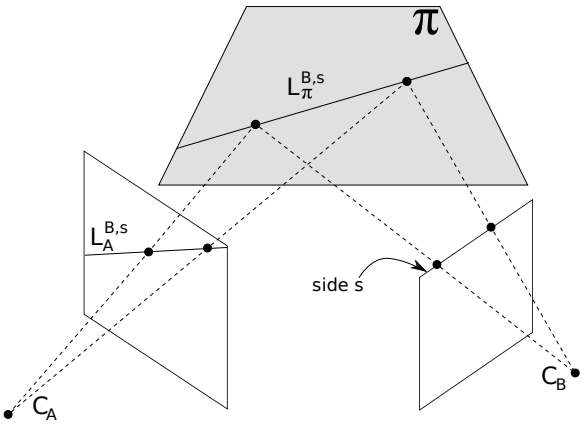


Figure 2.2: This thesis and [24] find field of view lines on ground plane π.

There are other approaches that effectively recover a geometric relationship between cameras. Starting from an appearance-based matching

method, [26] uses a homography-based method to recover from occlusions during which the appearance model can not be updated. The user effectively specifies the homography. Point intersections are used in [27] to create a 3D map of the world. Correct point intersections reinforce each other, leading to higher probability of a match, while incorrect matches are more scattered around the world, and so lead to a lower probability of matching.

Chapter 3

Proposed method

3.1 Overview

As described in Sections 1.2 and 1.3, an algorithm is desired that receives two video camera feeds from camera A and camera B. The cameras have at least partially overlapping fields of view. By watching people move through the scene the algorithm should eventually be able to predict the location of a person in camera B based solely on their position in camera A, and vice versa. This thesis will implement three functional objectives:

- Single-camera tracking,
- Learning the relationship between two cameras, and
- Using the relationship to predict target locations.

In Section 3.2, various algorithms used in this thesis will be explained. First, the two parts of single-camera tracking – background subtraction and object tracking – will be discussed. This will be followed by the method used to learn the relationship between camera A and camera B, which consists of two main parts: finding field of view lines and then accumulating enough corresponding points to calculate a plane-induced homography. Finally, once that homography is known, the novel algorithm that creates, updates, and refines relationships between targets will be introduced in Section 3.2.7.

Following the algorithm development, Section 3.3 will cover how the various components of the system will be tested.

3.1.1 Notation

In general, the following conventions will be used except when we adopt the notation from specific papers:

- Scalars are represented with lower-case letters. $y = 3$.
- Vectors are represented with bold lower-case letters, and are in column form. Transposing a vector gives us a list made up of a single row. $\mathbf{x}^T = [x_1\, x_2\, x_3]$.
- Matrices are represented with upper-case letters. In $\hat{\mathbf{x}} = H\mathbf{x}$, $\hat{\mathbf{x}}$ and $\mathbf{x}$ are column vectors, and H is an appropriately-sized matrix.

3.2 Algorithms

3.2.1 Background subtraction

The first step in processing each single-camera input frame is to determine which parts of the image are background, and hence un-interesting, and which are foreground. In this work, foreground objects are moving people. The algorithm first described in [28] and [29], and elaborated on in [30], describes a system that accomplishes these goals. Although the system is designed to work on traffic scenes, where the targets (also known as Moving Visual Objects, or MVOs) are cars, the algorithm has the following desirable attributes:

- The constantly-updated background model can adapt to changing lighting conditions, objects becoming part of the scene background (*e.g.* parking cars), and formerly-background objects leaving the scene (*e.g.* departing cars);
- Shadow-suppression; and
- Parameters to tune for various object speeds and sizes.

The general flow of the algorithm can be seen in Figure 3.1.

The background model is made up of a number of the previous frames, held in a FIFO queue. A number of methods can be used to initialize the buffer. For instance, the first frame read can be replicated to fill each

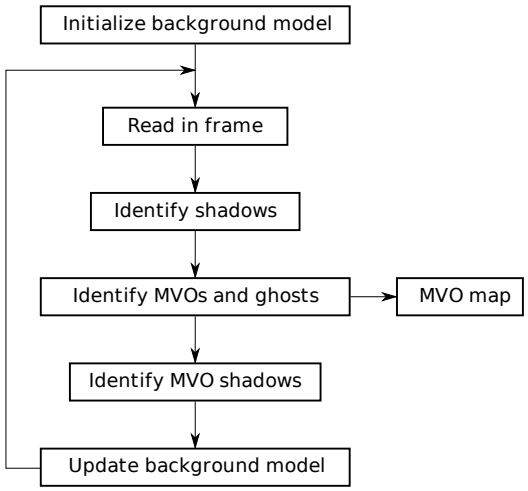


Figure 3.1: The background subtraction algorithm from [30].

slot in the queue, although if the frame has foreground elements then the MVO map will be incorrect at the start of processing. Alternatively, the algorithm can be set to run in a training mode, reading in, say, 100 frames. The median of those frames can then be calculated and the result used to fill the background model queue. The number of frames used to generate this model is a trade-off between the amount of memory available and the amount of traffic in any particular pixel. The method to calculate the background image from the buffer is described below.

In the main loop, there is a background image B. Any particular pixel, p, in the background has a hue $B_H(p)$, saturation $B_S(p)$, and value $B_V(p)$, which are easily calculated from the pixel's red, green, and blue values.

An image, I, is read from the incoming video stream. Pixel p in the incoming image has hue, saturation, and value $I_H(p)$, $I_S(p)$, and $I_V(p)$.

Once an image has been read, the first step is to identify shadow pixels. Pixel p in the shadow map, SM, is classified as shadow if:

$$SM(p) = \begin{cases} & \alpha \leq \dfrac{I_V(p)}{B_V(p)} \leq \beta \\ 1 \quad \text{if} & \mid I_S(p) - B_S(p) \mid \leq \tau_S \\ & min\left(\mid I_H(p) - B_H(p) \mid,\ 2\pi - \mid I_H(p) - B_H(p) \mid\right) \leq \tau_H \\ 0 \quad \text{otherwise} & \end{cases} \tag{3.1}$$

The first condition lets pixels be shadow if they have a value ratio between two thresholds. In [29] a preliminary sensitivity analysis was carried out, with values of $\alpha \in [0.3, 0.5]$ and $\beta \in [0.5, 0.9]$.

The next two conditions essentially require that the image pixel have the same colour as the background. Thresholds τ_S and τ_H are set by the user. In the analysis mentioned above, $\tau_S \in [0.1, 0.9]$ and $\tau_H \in [0, 0.1]$.

Once the shadow map has been created, MVOs are identified. First, every image pixel's distance from the background is calculated. The distance measure used is the largest absolute difference from the background of each of the pixel's hue, saturation, and value.

$$DB(p) = max_{H,S,V}\left(\mid I_{H,S,V}(p) - B_{H,S,V}(p) \mid\right) \tag{3.2}$$

Pixels with $DB(p)$ lower than a threshold T_L are not given further consideration as foreground. A morphological opening is performed on the blobs of pixels with $DB(p) > T_L$. Following the opening, any pixels that were previously found to be shadows are turned off, not to be considered as potential foreground pixels. Next, each remaining blob is checked for three conditions. If the blob satisfies all three conditions, then its pixels are labelled as part of an MVO.

- The blob has an area larger than threshold T_{area}.
- The blob contains at least one pixel with $DB(p) > T_H$. Note that $T_H > T_L$.
- The blob's average optical flow must be above threshold T_{AOF}.

Objects that do not meet all three criteria are called ghosts. No guidance is given in [30] on how to select the thresholds T_H, T_L, T_{area}, or T_{AOF}. Specific implementation details used in this thesis are discussed in the Chapter 4.

At this point, the MVO map has been created, and can be passed to the single-camera tracking algorithm. However, the background model must still be updated for the next frame. To do this, the shadow map is partitioned into foreground shadows and ghost shadows. Shadow pixel blobs that touch any of the foreground blobs are, naturally, labelled as

foreground shadows. Ghost shadows are pixels in shadow blobs that do not touch an MVO.

There are now five possible classifications for any given pixel: foreground MVO, foreground shadow, background, ghost, and ghost shadow. The background model update proceeds as follows:

1. Pixels in the current frame that have been marked as foreground or foreground shadow are replaced with the pixels from the current background image.

2. The modified current frame is pushed onto the background FIFO queue. The oldest image, at the end of the queue, is discarded.

3. n images are selected from the buffer, equally spread by m intervening frames. For instance, every fifth image might be selected ($m = 5$), with eight images in total ($n = 8$). The queue is sized $n \times m$.

4. The median of the selected is calculated. The result is the updated background image for the next incoming frame.

3.2.2 Single-camera tracking

In order to associate targets using multiple cameras, we first need to track moving objects in a single camera. The input to the single camera tracker

is the output of the background subtraction algorithm, namely, a series of blob masks that is non-zero on foreground pixels, and zero elsewhere. The output of the single-camera tracker can be represented many ways. In the present work the output is a frame-sized array, with each pixel's value an integer corresponding to a blob label. As targets move around the scene the shape and position of their corresponding blob changes with the output of the background subtraction algorithm. After single-camera tracking, the value of the pixels in each target's blob should be constant.

We add the additional requirement that target labels should be historically unique. This means that once a target disappears from the scene, whether by leaving the field of view of the camera or by being fully-occluded, its tracking label is not used again.

Depending on the machine architecture, the language used, and the expected run times it is possible that the target label variable may eventually overflow. If this event is expected to occur inside the temporal window where the oldest target and the newest target are active in some part of the surveilled area, then there might be a problem. Since the scenes used in this research were fairly short and had low traffic rates, we did not encounter this problem when using 16-bit unsigned integer target labels (65,535 possible labels).

The single-camera tracking algorithm used in this thesis can be found in [5]. For basic motion it uses a simple mask-overlap method of track-

ing blobs from frame to frame. In more complex motion cases, such as partial occlusion, merging, and splitting, the algorithm uses a shape-matching technique developed in [6]. The following is a brief overview of the single-camera tracking algorithm; for more detail the reader is encouraged to examine the original papers.

Overlap tracking

In an un-crowded scene, targets move around without being occluded by other moving targets. In this case tracking is a fairly simple process if one key assumption is made: slow motion. This means that targets are required to be imaged in such a position that they partially overlap their previous location. For most surveillance applications with reasonably-sized targets and normal video frame rates, this is a reasonable assumption.

Given a map of foreground pixels identified by the background subtraction algorithm for frame t, and the output of the single-camera tracker from the previous cycle (frame $t-1$), we perform the following tests:

- Does each blob in frame t overlap any previously-identified targets in frame $t-1$? If so, how many?
- How many blobs in frame t overlap each target from frame $t-1$?

If the answer to the first question is zero then the blob is a new target.

It is given a new target label, and the new target counter is incremented.

If the answer to the both questions is exactly one then the slow motion assumption means that the blob *is* the same target as the one that it overlaps. Thus, the blob in frame t inherits the target label from frame $t-1$, and for that blob, tracking is complete.

Complex motion – splitting

If two or more unconnected blobs in the current frame t overlap one contiguous target in frame $t-1$, then this indicates a splitting event. This could happen when two people who originally walked side-by-side take different paths.

In this case, [5] says that the largest blob in frame t should inherit the tracking label from frame $t-1$, and the smaller blob(s) should receive a new, unique tracking label(s).

Complex motion – merging

If a contiguous blob in frame t overlaps more than one previously identified target region then we must determine which pixels in the blob belong to each of the old targets. This situation occurs whenever two targets walk towards each other – there is a period of time when one target partially occludes the other. Matching is done using the shape matching technique introduced in [6]. Essentially, the algorithm proceeds as follows:

1. The current frame is segmented into regions based on colour and texture information using the algorithm found in [7].

2. B-splines are fitted around various segments, starting with the segment that has the largest area. The splines are fitted according to algorithms found in [31].

3. The B-spline control points of the targets in frame $t-1$ are compared to the control points of shapes made up of the frame t image segments using a metric based on the Hausdorff distance. The metric is invariant to affine transformations [6].

4. The segments that make up the best match for each of the previous target shapes inherit the old tracking labels.

Note that if one target fully occludes the other then the occluded target's tracking label is deprecated and is not used again. When the occluded target emerges from full occlusion back into partial occlusion the tracker essentially believes that the occluding target is growing. Eventually the targets split into two non-contiguous blobs, and are treated as a described in the splitting section, above. Thus, when a target passes through full occlusion, it will emerge with a new target label not related to its old label.

3.2.3 Field of view line determination

Finding the relationship between the objects in camera A and objects in camera B can be broken into two parts. The first part consists of finding Field of View (FOV) lines and creating target associations using a modified implementation of [24]. That algorithm shall be described in this subsection. The second part consists of using the meta-target information to create a list of corresponding points (Section 3.2.5), and then using those points to find the plane-induced homography (Section 3.2.6).

As discussed in the previous chapter, the aim of [24] is similar to the aim of this work. In short, we want to find the relationship between moving people as seen by cameras A and B. Whereas this thesis's approach wishes to find the correspondence between targets at any point in the frame, the FOV approach is designed to create meta-targets based on a single correspondence event between tracks of moving objects. This single correspondence event occurs when the target passes into or out of the field of view of camera B. That instant is known as an "FOV event". In [25] an FOV event is slightly redefined to be the time at which an object has completely entered a scene, and is not touching any frame edge. FOV events are also triggered at the instant when an in-scene target begins to touch a frame edge. In the code implemented for this thesis we use the modified definition of an FOV event. Figure 3.2 shows a target just as it is about to trigger an edge event.

In the following paragraphs we use notation from [24].

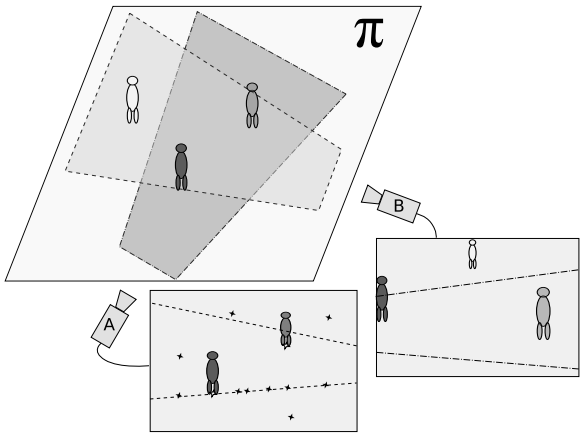


Figure 3.2: In camera B a target is almost completely in the field of view. When no part of the target mask touches the frame's edge then an FOV event is triggered.

Once an FOV event has been triggered by object k as seen in camera B, O_k^B, the FOV constraint allows us to choose the best matching object O_i^A from the set of all moving objects in camera A, O^A. We quote from [24]:

> If a new view of an object is seen in [camera B] such that it has entered the image along side s, then the corresponding view of the same object will be visible on the line $L_A^{B,s}$ in [camera A]

The line $L_A^{B,s}$ is the field of view line of side s of camera B, as seen by camera A. This is illustrated in Figure 3.3.

The algorithm essentially has two steps: learn the FOV lines of camera B as seen by camera A, and then create associations whenever a target

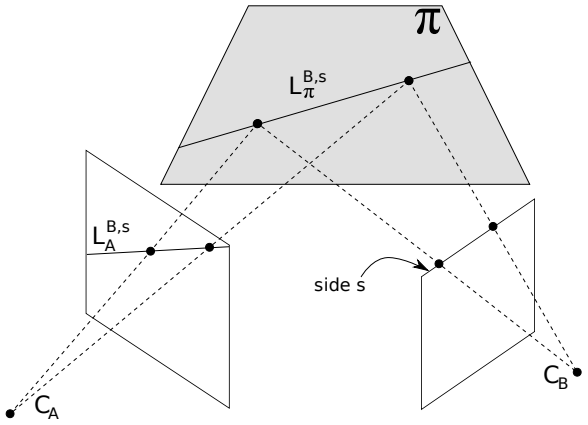


Figure 3.3: The top side of camera B projects to a line on ground plane π, shown as $L_\pi^{B,s}$. That line is then imaged by camera A as $L_A^{B,s}$.

steps into camera B. Automatic recovery of the FOV lines is done by following these steps:

1. Whenever a target enters or leaves camera B on side s, trigger an FOV event.

2. When an FOV event has been triggered, record the feet feature locations of all targets in camera A. Note that each side of camera B will have its own map of accumulated feet locations.

3. After a minimum number of FOV events have been recorded, perform a Hough transform on the map of feet points. The result is the FOV line in camera A, $L_A^{B,s}$.

Thereafter, whenever a target crosses side s of camera B, triggering an FOV event, it can be associated with the target in camera A that is closest

to $L_A^{B,s}$. To find the target with the feet closest to the FOV line, we can use the homogeneous representation of $L_A^{B,s}$: $\mathbf{l} = [a\ b\ c]^T$ with $ax + by + c = 0$, scaled such that $a^2 + b^2 = 1$. x and y can be taken as positions on the row and column axes of camera A. If the homogeneous representation of the target's feet point is used, $\mathbf{x} = [x\ y\ 1]^T$, then the perpendicular distance from the line to the feet point is given by the dot product $d = \mathbf{l}^T\mathbf{x}$.

The initial association of targets as they cross FOV lines essentially creates a point correspondence between the two feet feature locations. Khan and Shah say that they can use those point correspondences to find a homography between the two cameras [24]. Once the homography is known, they can use it to predict target locations, thereby creating target associations for targets that are not close to a FOV line. Although a projective transformation was noted to be the general solution, [24] noted that an affine transform was sufficient for their purposes. No details are given on how to implement this homography-based matching system. If only one side is used by targets to enter and exit the frame then the homography will be degenerate, and the matching system will not work.

The method described in [25] also attempts to find a homography using point correspondences. However, instead of relying on target-created point correspondences, they use the four points created by intersecting FOV lines. This method requires that all four FOV lines be known. If one or more edges have insufficient traffic to find the FOV line, then the ho-

mography is not computable with this method.

The method of [25] also introduces a potential source of error: because FOV events are triggered when the target is fully inside the frame, the camera B FOV lines are displaced towards the centre of the frame. Yet the corner points, used for the calculation of the homography, are left at the corners of the frame. Because the homography only uses four points, it is exact (*i.e.* it is not a least-squares error-minimizing solution with many input points), so it will certainly contain errors. The exact nature of the error will depend on the angle of the cameras, the average height of segmented targets, and which FOV lines are inaccurate.

In relying solely on the FOV line point correspondences, [24] effectively makes the following assumptions:

1. At the moment they cross an FOV line, all targets are properly segmented in both viewpoints.

2. The location of the feet feature is correctly determined for all targets.

3. A homography based on points near FOV lines will predict feet locations for targets in the middle of the frame with sufficient accuracy to associate targets from camera to camera.

Ideally, these restrictive assumptions will be valid. However, if the segmentation of targets is incorrect in either camera, either through addition (*i.e.* background pixels wrongly flagged as part of an MVO) or subtraction

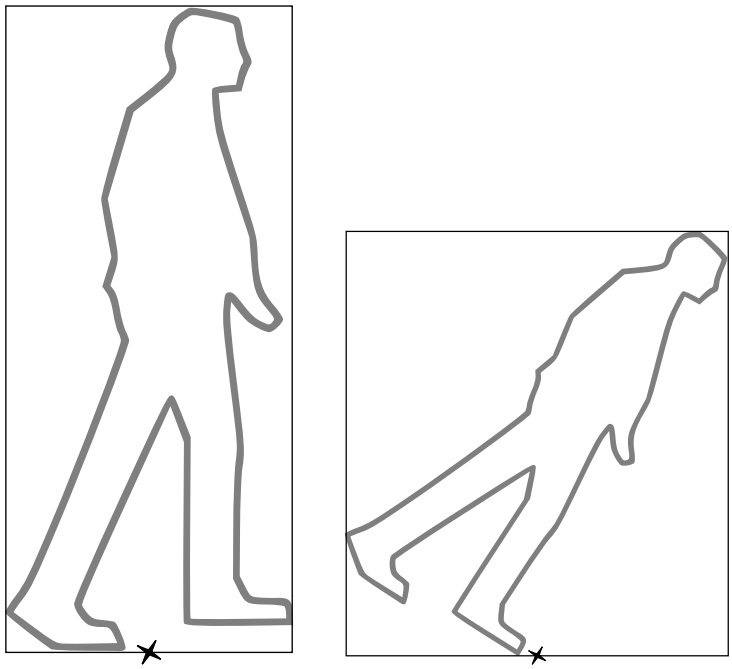
(*i.e.* MVO pixels not included in the target's mask), then the feet locations will be incorrectly identified. This will have many knock-on effects: the FOV lines will be skewed; a target may be incorrectly judged as being closest to the FOV line; the list of corresponding points will be incorrect; and thus the homography used for matching might contain significant errors.

We overcome these potential sources of error using methods introduced below in Sections 3.2.5 and 3.2.7. First, we discuss two ancillary algorithms: how to find the feet feature, and how to calculate a homography given a list of corresponding points.

3.2.4 Determining feet locations

Finding the feet of a moving object is essentially a feature detection problem. In the present case, the input to the feet locator function will be a mask of a moving object – a standing or walking person. The challenge is to find a single point that represents the "feet" of that person. In general, a person will have two feet touching or nearly touching the ground. The single point then can be thought of as representing the location of the person's centre of gravity, projected onto the ground plane.

As mentioned above, the requirement for a single point stems from our desire to project that point into another camera's coordinate system. The projection of a person's centre of gravity should be onto the same point on the world plane regardless of the location of the cameras.



(a) The feet are said to be at the centre of the bottom edge of the bounding box.

(b) If a target is tilted relative to the camera, then the method will fail to correctly identify the feet location.

Figure 3.4: The method of finding the feet feature location from [24].

In [24], Khan and Shah drew a vertical rectangular bounding box around a moving object. The target's feet were then deemed to be positioned at the centre of the bottom edge of the bounding box. This can be seen in Figure 3.4(a). This method is widely used in the computer vision community.

The bounding-box method of finding feet described in [24] has a significant potential failure mode. It effectively requires the person to have

their height axis aligned with the vertical axis of the camera. If the camera is tilted relative to the person, which could occur if the camera is installed on a tilted platform or if the person has a consistent tilt in their gait, then the bounding box will enclose a large amount of non-target area. As a result, the bottom centre of the bounding box will not necessarily be close to the actual feet of the target. Figure 3.4(b) shows this error mode.

The second failure mode of the bounding-box feet-finding method is that the feet will always be outside of the convex hull formed by the target mask. This means that the feet will always appear below the actual feet location. When considered in three dimensions for targets on a ground plane, this means that the feet will always be marked closer to the camera than they should be. Given a wide disparity in camera views, the projected locations of the feet in each view will not be at the same point – the feet will be marked somewhere between the best feet location and the cameras' locations.

Instead of relying on the camera and the target's height axis to be aligned, if the target is rotated to a vertical position then the problem of tilted targets is eliminated. One of many methods to find the rotation angle is to perform a principle components analysis (PCA) on the target mask. On a two-dimensional mask, the output of the PCA can be interpreted as a rotation that overlays the direction of maximum variability with the first axis, *i.e.* the row axis. For most camera angles, people will

appear to be much taller than they are wide, so the direction of maximum variability will be the height axis.

Once the PCA has rotated the object to eliminate the effect of a tilted target, the actual position of the feet needs to be found. Instead of simply using the centre of the bottom of the rotated bounding box, which may lead to an inaccurate feature location as described above, the following method can be employed:

1. Trace the outline of the target mask.
2. For each point on the outline of the mask, find the distance to the two bottom corners of the bounding box.
3. Find the point with the smallest distance to each of the two corners.
4. The feet are located halfway between those two points.

The method is illustrated in Figure 3.5.

Height determination

Section 3.2.5, below, requires the measurement of the height of a target. This is a solved problem for calibrated cameras: [32] showed a method to find the actual in-world height of an target. However, this thesis does not use calibrated cameras. Luckily, the algorithm does not need the real-world height – only the target's apparent height in pixels is needed.

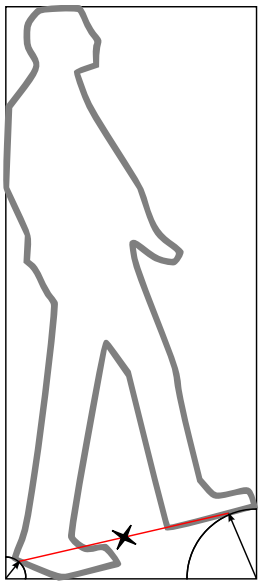


Figure 3.5: After using PCA to align the height axis with the bounding box, the two points on the target's outline closest to the bounding box's corners are averaged. The result is the location of the feet feature.

The method to determine the height begins similarly to that described when finding the feet above. Again, since a target may be rotated with respect to the camera's vertical axis, we can not simply take the height of the bounding box as the height of the object. Rather, the PCA algorithm is applied to the pixels in the target mask, and the target is rotated upright. Now the vertical axis is parallel with the target's height axis. At this stage there are two options, simply take the height of the bounding box as the height of the target, or find the feet feature and take the vertical distance from that point to the top of the bounding box as the height. The latter method, which can be seen in Figure 3.6, is used in this thesis.

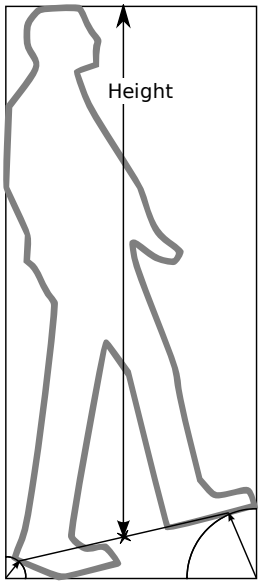


Figure 3.6: The height of a target is the vertical distance between the feet and the top of the correctly-oriented bounding box.

In contrast to the feet, the head of a target does not split into two elements, so the top of the bounding box is nearly always at the correct location on the target's head. Furthermore, the shoulders and arms are often closer to the bounding box corners than the closest head pixel. This justifies not using the same feet-finding method on the head, and instead simply using the top of the bounding box.

3.2.5 Dropping markers

In Section 3.2.3 we noted that [24] used only corresponding point pairs that were found during FOV events to calculate the plane-induced homog-

raphy H_π. These points naturally line the edges of the frame of camera B. However, since only one point is created for each target (unless the target surfs the FOV line), the correspondence points will be sparse. Depending on where targets enter and exit the frame, there may be many collinear correspondences on one FOV line and only a few on the other lines. If only one edge serves as an entry and exit point, then all the accumulated feet locations will be collinear, and the homography will be degenerate for our purposes.

Another method was introduced in [25]: using exactly the four points that are formed by intersecting the FOV lines and the corners of the frame of camera B. This method has a large potential pitfall: if one or more FOV lines are unknown, then the method can not work. It is trivial to find a situation where an FOV line can not be found – anytime camera B has one edge above the vanishing line of the plane formed by the targets' heads (*i.e.* slightly above the horizon), no targets will trigger edge events on that edge. Therefore, that FOV line will never be found, and the homography will never be computable.

An improved method to generate the homography is desired. Ideally, the method will have the following attributes:

- Many corresponding points,
- Non-sparse corresponding points, and

- Able to create corresponding points even if all targets enter and leave the scene through the same edge.

In [33], Hänsel and Gretel drop a trail of bread crumbs behind them as they travel through the woods. We adapt this idea of a trail of bread crumbs to the present algorithm. In this case, the bread crumbs, also called markers, are located at the feet feature of each target, which is assume to be a point on the ground plane π. This can be seen in Figure 3.7.

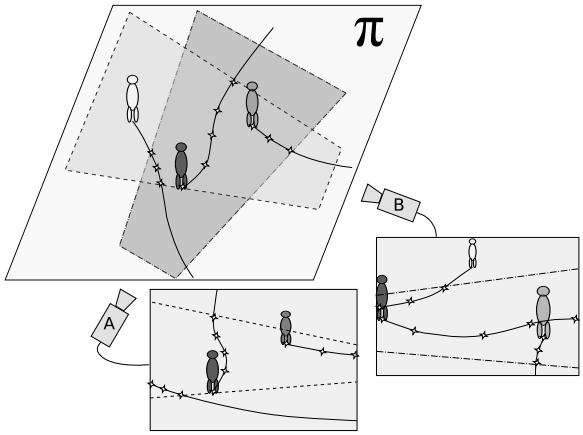


Figure 3.7: As targets move through the scene, they leave trails of markers behind them. These markers are pairs of point correspondences.

After a meta-target is created by associating a target from camera A with a target from camera B, at an FOV event or using the homography-based method described in Section 3.2.7, the marker-dropping algorithm begins. The logical flow through the algorithm is shown in Figure 3.8. Every frame, a number of tests are performed to decide whether to drop a

marker. If the tests are passed then a marker is dropped, thereby creating a corresponding point pair.

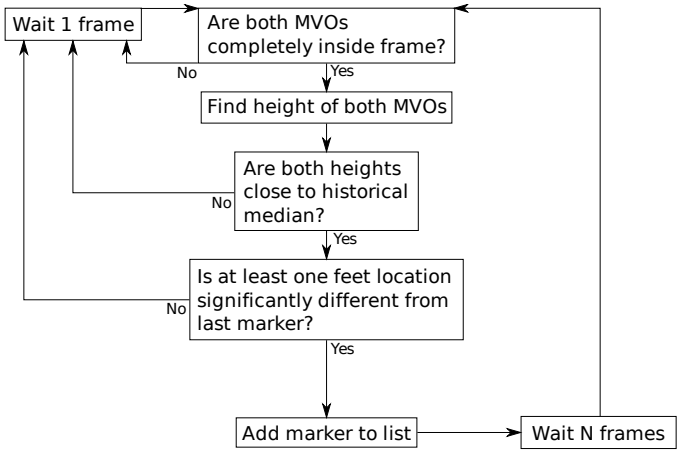


Figure 3.8: The flow of the corresponding-point generation algorithm. After creation, this algorithm is run on every meta-target.

The first test simply detects whether both target masks are completely inside their respective frames. If one target is even partially out of the frame then it is impossible to say with certainty where the feet feature is located. Therefore, it is a bad idea to create a corresponding point pair.

The second test is of the height of the target. This test is designed to prevent a marker from being dropped if in that particular frame the target is grossly mis-segmented or partially-occluded. The method used to determine the target height was outlined in Section 3.2.4. The height of an object is measured in pixels, and compared to the median of the heights in the past few frames. If the height is similar, then we pass on to the next test. If it is significantly different, we abort processing of that

meta-target until the next frame. This test should work because the height of a target should not change radically over the course of a few frames. So long as the threshold is well-chosen, this test should only throw away badly-segmented or partially-occluded targets.

The third test is to determine whether there has been significant motion in the past few frames. The distances of the meta-target's feet features from the last marker location are calculated. One of the following three cases must be true:

- Both feet features are close to the previous marker. This means that the target has not moved much since dropping the last marker. Creating a new marker will essentially duplicate the previous marker, so it is not useful. No marker should be dropped in this case.

- Exactly one of the targets has moved away from the previous marker. This could have two explanations.

  The first possibility is that the target has moved towards or away from one camera centre. In that camera the target will appear in nearly the same location. However, in the other camera the target might be seen to move across-frame. This motion is important in the creation of the homography, so the marker should be dropped.

  The second explanation is that the target was segmented as foreground significantly differently in the current frame than when the

last marker was dropped, but it still passed the height test. In this case adding the marker to the list will will likely increase the accuracy of the homography. This is because on average we expect the feet feature to be correctly identified, so any incorrect feet will be drowned out in the least-squares nature of the DLT algorithm.

- Both targets have moved away from the previous marker. This probably indicates real target motion. In this case a marker should be dropped.

One of the requirements stated in Section 1.3 was that neither camera can be on the ground plane. If the camera is in the ground plane then all of the markers in that camera will appear on a line – the line formed by the intersection of the camera's focal plane and the ground plane. This geometry forces the markers to be collinear, which leads to a degenerate homography.

The end result is a list of corresponding points. If plotted on the frame, the points will follow show the historical tracks of the meta-targets. Depending on meta-target movement – how people walk through the environment – the points may initially be roughly collinear. A homography calculated based on corresponding points when one set of points is collinear will be degenerate. Therefore, before calculating the homography, we must test for collinearity.

3.2.6 Calculation of a homography

Given a point that lies on a world plane π, $\mathbf{X} = [X_1\ X_2\ X_3\ 1]^T$ we wish to find a predictive relation between the images of $\mathbf{X}$ in two cameras. That is, given image points $\mathbf{x} = [x_1\ x_2\ 1]^T$ and $\mathbf{x}' = [x_1'\ x_2'\ 1]^T$, we wish to find H_π such that

$$\mathbf{x}' = H_\pi \mathbf{x} \tag{3.3}$$

H_π is said to be the homography *induced* by the world plane π. It can be thought of as two projectivities chained together: one that takes a two-dimensional point from the image plane of the first camera, $\mathbf{x}$, to a two-dimensional point on plane π, and a second that takes a point from plane π to a point on the second camera's focal plane, $\mathbf{x}'$. (Note that a point that lies on a plane in the three dimensional world only has two degrees of freedom – those that are required to move on the two dimensional plane.) The geometry of such a scene can be seen in Figure 3.9.

Calculation of H_π is a fairly simple using the Direct Linear Transformation (DLT) algorithm. We follow the notation used in [34]. First, given $n \geq 4$ corresponding points of form $\mathbf{x}_i = [x_i\ y_i\ w_i]^T$ and $\mathbf{x}_i' = [x_i'\ y_i'\ w_i']^T$,

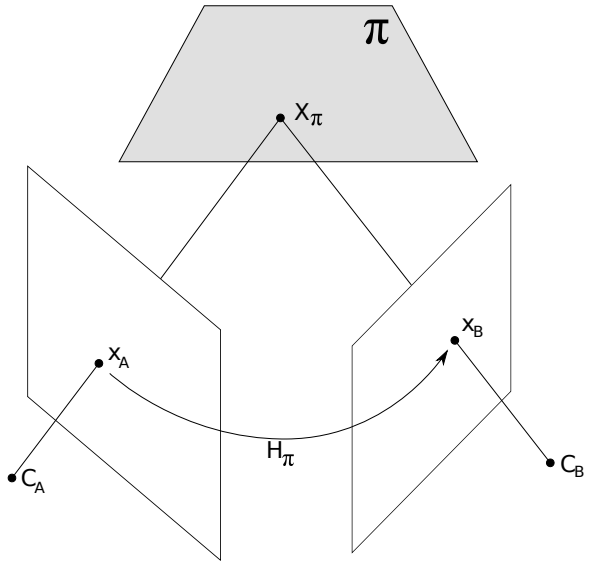


Figure 3.9: A world plane π induces a projective homography between the two image planes. $\mathbf{x}_B = H_\pi \mathbf{x}_A$. After [34] Fig. 13.1

we can take the cross product of Equation 3.3:

$$\mathbf{x}'_i \times H_\pi \mathbf{x}_i = \mathbf{x}'_i \times \mathbf{x}'_i = \mathbf{0} \tag{3.4}$$

If expanded, the cross product can be written explicitly as a matrix multiplication with the elements:

$$\begin{bmatrix} \mathbf{0}^T & -w'_i\mathbf{x}_i^T & y'_i\mathbf{x}_i^T \\ w'_i\mathbf{x}_i^T & \mathbf{0}^T & -x'_i\mathbf{x}_i^T \\ -y'_i\mathbf{x}_i^T & x'_i\mathbf{x}_i^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{h}^1 \\ \mathbf{h}^2 \\ \mathbf{h}^3 \end{bmatrix} = A_i\mathbf{h} = \mathbf{0} \tag{3.5}$$

where H has been rearranged into a column vector $\mathbf{h}$ using a lexico-

graphic ordering:

$$H = \left[\mathbf{h}^1 \; \mathbf{h}^2 \; \mathbf{h}^3\right] \tag{3.6}$$

In Equation 3.5, it can be seen that the third row is actually a linear sum of the first two rows. Therefore, there are only two linearly independent rows in A_i. This makes intuitive sense. Each point correspondence provides two constraints on the homography. The third element in each homogeneous point, w_i and w_i', are simply scale factors that do not constrain H.

Therefore, if we only write the first two rows of A_i for each of the $n \geq 4$ point correspondences, then we can stack each A_i into a $2n \times 9$ matrix A. Assuming that we have some measurement noise in our pixel locations, then the solution to $A\mathbf{h} = \mathbf{0}$ will not be exact. We therefore attempt to minimize $\|A\mathbf{h}\|$. As shown in [34], finding the solution that minimizes $\|A\mathbf{h}\|$ is equivalent to minimizing $\|A\mathbf{h}\|/\|\mathbf{h}\|$.

The solution is the unit singular vector corresponding to the smallest singular value of A. By taking the Singular Value Decomposition (SVD) of A, such that $A = UDV^T$, $\mathbf{h}$ is the column in V corresponding to the smallest singular value in D. H is obtained by simply rearranging the values of $\mathbf{h}$ into a 3×3 matrix.

Data normalization

In [35], Hartley noted that many users of the 8-point algorithm did not pay close attention to numerical considerations when performing their calculations. The 8-point algorithm is used to find the essential matrix, but the first steps of the algorithm are very similar to the DLT described above.

The arguments in the 1995 paper, re-explained in [34], boil down to this: because x_i and y_i are typically measured in the hundreds, whereas w_i is usually about 1, some entries in A_i will have a vastly different magnitudes then others. When the SVD is performed and $\mathbf{h}$ is found, the effects of the low-magnitude entries of A will be drowned out by the high-magnitude entries, and the result will be inaccurate.

The solution is to normalize or pre-condition the input data $\mathbf{x}_i$ and $\mathbf{x}'_i$. The normalization of each data set is performed by calculating two 3×3 transformations, T and T', that

1. Translate the points such their centroids are at the origin, and

2. Scale the points such that the mean absolute distance from the origin along the x and y axes is 1.

The second step is equivalent to scaling the points such that mean absolute distance from the origin is $\sqrt{2}$.

The two transformations are applied to the two data sets before the DLT algorithm is performed. After transformation, the average absolute values of x_i, y_i, and w_i will all be 1, and the solution will be numerically well-conditioned.

Thus, instead of using $\mathbf{x}$ and $\mathbf{x}'$, the inputs to the DLT are $T\mathbf{x}$ and $T'\mathbf{x}'$. The output of the DLT will not operate directly on the image coordinates – H_{DLT} will only be valid for normalized coordinates. To directly use image coordinates, we observe that:

$$\begin{aligned} T'\mathbf{x}' &= H_{DLT}T\mathbf{x} \\ \mathbf{x}' &= T'^{-1}H_{DLT}T\mathbf{x} \\ \mathbf{x}' &= \left(T'^{-1}H_{DLT}T\right)\mathbf{x} = H_\pi\mathbf{x} \end{aligned} \tag{3.7}$$

In other words, H_π is the product of three matrices:

- A normalization transformation that works on points in the coordinate system of the first image,
- H_{DLT}, the output of the DLT algorithm fed with normalized data, and
- The de-normalization transform that takes normalized coordinates into the coordinate system of the second image.

It should be noted that the normalization step is very important. Although the algorithm will appear to work correctly, without normalization the predicted locations $\mathbf{x}'$ will be incorrect. Not including the data normalization step is an insidious error.

3.2.7 Multiple-camera tracking with a homography

Let us re-capitulate the algorithm to this point. Two cameras' video feeds were taken, background subtraction was performed, and unique tracking labels were assigned to each target in each camera. By watching for Field of View (FOV) events, the lines marking the edges of the field of view of camera B, as seen by camera A, were found. As a target crosses those lines, it is associated with a target in the other camera. These associations are called meta-targets. As the meta-targets move around the world they leave a trail of markers behind them, creating a list of corresponding points on the world plane. Once the list of points in both cameras is sufficiently non-collinear, the Direct Linear Transform (DLT) algorithm calculates the homography, H_π, induced between the two cameras by the world ground plane.

The task now is to take H_π and the list of meta-targets found using FOV lines, refine the list, and correctly identify any meta-targets in the scene. This process continues *ad infinitum*.

We first use the algorithm of Section 3.2.4 to find the feet feature loca-

tion for each active target in both cameras. The projected location of each camera A target in the camera B coordinate system is given by

$$\hat{\mathbf{x}}_A = H_\pi \mathbf{x}_A \tag{3.8}$$

The inverse projection can also be carried out. The feet location of each target in camera B projects to a location in the camera A coordinate system by:

$$\hat{\mathbf{x}}_B = H_\pi^{-1} \mathbf{x}_B \tag{3.9}$$

In both of the previous equations, the hat (circumflex accent) signifies that the location has been projected into the other camera's coordinate system and that it is not directly measured.

If the projected location of a target is within the bounds of the frame, then the distance is calculated from that projected location to the feet of all targets in that frame. The distance measure used is the scaled symmetric transfer distance (SSTD). We use the two terms interchangeably. For a given homography H_π, a point $\mathbf{x}_A$ in camera A, a point $\mathbf{x}_B$ in camera B, and the diagonal size in pixels of the two frames, d_A and d_B, we define the scaled symmetric transfer distance ϵ as

$$\epsilon = \frac{\|H_\pi \mathbf{x}_A - \mathbf{x}_B\|}{d_B} + \frac{\|H_\pi^{-1} \mathbf{x}_B - \mathbf{x}_A\|}{d_A} \tag{3.10}$$

$$= \frac{\|\hat{\mathbf{x}}_A - \mathbf{x}_B\|}{d_B} + \frac{\|\hat{\mathbf{x}}_B - \mathbf{x}_A\|}{d_A} \tag{3.11}$$

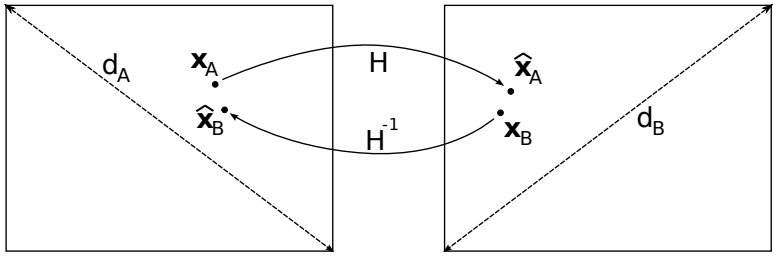


Figure 3.10: The scaled symmetric transfer distance is the sum of the distances between the points and their projected cousins, each divided by the diagonal frame size.

The terms are shown graphically in Figure 3.10. The SSTD represents a frame-size-independent method of determining which targets are closest to each other. In the very worst case, with the homography projecting the points to opposite corners from the real points, the SSTD will have a maximum value of 2. A flowchart depicting the first part of the multiple-camera tracking algorithm is shown in Figure 3.11

Following calculation of each of the applicable distances, the pair of targets corresponding to the smallest distance is found. If that smallest distance is above some maximum meta-target creation distance threshold τ_ϵ then no more good matches can be created and so the algorithm ends.

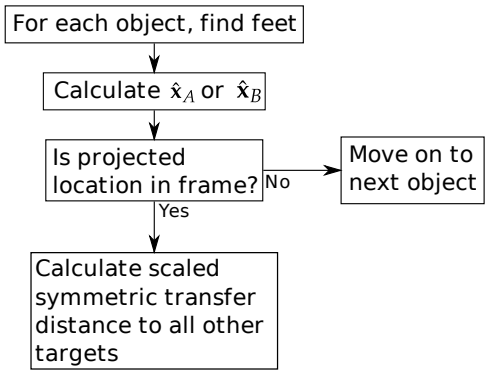


Figure 3.11: The first part of the algorithm consists of finding the SSTD for each valid pair of targets.

If the smallest SSTD is smaller than τ_ϵ then any of the following cases could be true of the pair of targets:

1. Neither of the targets is a member of an pre-existing meta-target. In this case a new meta-target is created to associate the two targets. This case occurs most frequently when a target appears in the middle of a frame, either through an entrance or after a full occlusion.

2. Both of the targets are members of the same pre-existing meta-target. This means that the current method agrees with the method used to create or maintain the meta-target, and that no changes need to be made to the meta-target.

3. Exactly one of the targets already belongs to a pre-existing meta-target. Should the pre-existing meta-target be replaced by the new pairing? First, the distance measure of the pre-existing meta-target

is retrieved. In theory, if the pre-existing meta-target's distance is larger than the current pair's distance, then we should delete the old meta-target and replace it with the new pair. However, this might introduce a flip-flopping behaviour where close real-world targets lead to meta-targets continuously being created and destroyed.

To combat this undesirable behaviour we introduce a change reticence threshold τ_Δ. In order to replace a pre-existing meta-target, the current pair of targets' SSTD must satisfy $\epsilon_{current} < \tau_\Delta \epsilon_{old}$. This means that a potential target pair will only replace a pre-existing meta-target if the relevant distance measurement is significantly better than the old distance.

4. Both of the targets belong to different pre-existing meta-targets. In this case the same logic is applied as if there was only one pre-existing meta-target. The pre-existing meta-target with the smallest SSTD is used for the comparison, *i.e.* $\epsilon_{current} < \tau_\Delta min\left(\epsilon_{old-1},\ \epsilon_{old-2}\right)$.

After the pair of targets has been dealt with the next smallest distance is found. The algorithm then repeats for the pair of targets corresponding to that distance, and so on until the smallest remaining distance is above the maximum threshold τ_ϵ. A flowchart depicting the second part of the multi-camera tracking algorithm is shown in Figure 3.12.

Any time a new meta-target is created, the row and column in the table

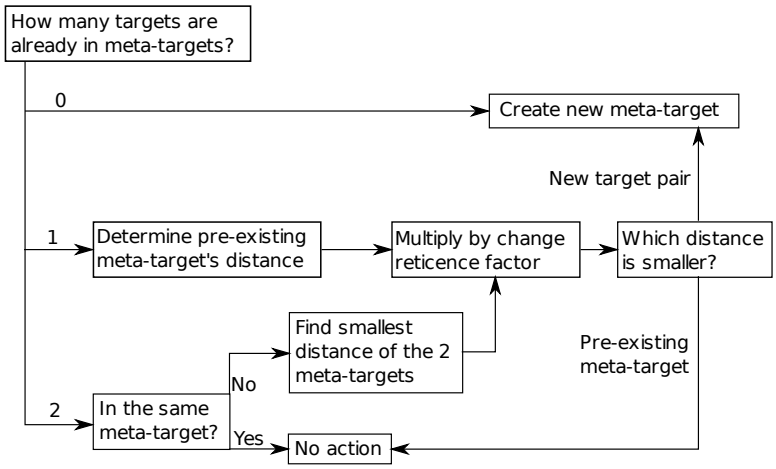


Figure 3.12: The pair of targets corresponding to the smallest distance is selected. After the steps shown here are complete, the pair with the next best distance is selected, and the flow repeats. This continues until the next smallest distance is above a threshold τ_ϵ.

of distances corresponding to the two targets are set to a large value. This prevents them from being considered further.

In the third and fourth cases listed above, there is a potential for members of a pre-existing meta-target to be orphaned by a newer and better meta-target. If this occurs, the orphan will necessarily have a larger distance to any other target. However, the next largest distance for the orphan may still be below the maximum meta-target creation distance threshold τ_ϵ. If this is the case then the orphan will be considered in one of the next passes through the distance matrix for re-assocation with a different target. Because the orphan will be considered in a later pass, no additional logic is required to deal specifically with orphans.

3.3 Testing and validation

The goal of the multiple-camera tracking system, stated simply, is to correctly associate target labels in one camera with target labels in another camera. Testing this might seem simple – look at each pair of targets when it is created, and divide the number of correct meta-target associations by the total number of meta-targets to get some measure of the accuracy of the algorithm.

Unfortunately, this simple approach only works for meta-targets created after the plane-induced homography H_π is known. In the short video sequences available to us, the time taken to learn H_π is variable and non-trivial. Indeed, since each new meta-target drops new markers as it moves, H_π is constantly changing. Rather than trying to measure the performance of the whole algorithm at once, we try to measure the performance of parts of the algorithm.

Our attention then turns to measuring the accuracy of the homography. The homography learned by the system must be able to accurately project points from one camera to the other. If this is not done well then invalid meta-target associations could be created, and valid meta-targets might not be found.

The system discussed above has had a couple of its components tested in other publications. Background subtraction was covered in [30]. The single-camera tracking system was developed and tested in [5]. Testing

of those components is not repeated in this thesis.

The critical parts of the algorithm left to be tested are the feet feature finder and the method to determine the homography, as well as the performance of the homography-based matcher.

3.3.1 Testing the feet feature finder

The feet feature finding algorithm looks at a mask of a target and returns a pixel location that best represents the centre of gravity of the person projected onto the ground plane. This task, tricky to do automatically by a computer, is simply and accurately performed by a human.

The algorithm shall be tested as follows:

1. At a proscribed interval in a video sequence, a frame shall be presented to a user with the target person indicated but the precise target mask not shown.

2. The user shall mark the location that best represents the feet feature. Because the target mask is not shown, it will not colour the user's perception of where the feet should be.

3. The algorithm shall mark the feet feature.

4. The Euclidean distance between the two locations shall be calculated.

5. The distance shall then be divided by the diagonal of the frame size, producing a scaled distance $\bar{d}$ as shown in Equation 3.12 below. If this step was not performed, then distances measured in a video camera with a large frame size would probably be larger than distances measured in a camera with a small frame size. Normalizing the distance in this fashion enables comparison of the performance of the algorithm when run on cameras with different frame sizes.

$$\bar{d} = \frac{\|\mathbf{x}_{human} - \mathbf{x}_{computer}\|}{d_{frame}} \tag{3.12}$$

If the feet feature finder works perfectly, $\bar{d}$ will be zero. The maximum possible value of $\bar{d}$ is 1, which would occur in the unlikely case that the user and the algorithm pick locations on opposite corners of the image. This is true regardless of the resolution of the camera, since the measure $\bar{d}$ is scaled by the frame's diagonal size.

For example, consider Figure 3.13. In this contrived example, the human selected the feet at $\mathbf{x}_{human} = [164\ 149]$ and the algorithm found the feet at $\mathbf{x}_{computer} = [153\ 171]$. The output of the camera is 240×360. Therefore,

$$\bar{d} = \frac{\|[164\ 149] - [153\ 171]\|}{\sqrt{240^2 + 360^2}} = 0.057 \tag{3.13}$$

In this case, $\bar{d} = 0.057$ indicates a difference of about 24.6 pixels in a

frame with a diagonal size of $d_{frame} \approx 433$ pixels.

Figure 3.13: In this contrived example, the green marker indicates the human-found location of the feet feature, and the blue marker indicates the location of the feet given by the algorithm. Note that this image is a blown-up section of the full frame.

This method can be used to compare the performance of the bounding box algorithm used in [24] to the performance of the rotated algorithm introduced in Section 3.2.4. The better algorithm will have a lower average value of $\bar{d}$. It is hoped that the rotated method will be generally better, and clearly better when the camera is tilted with respect to the height-axis of the targets.

3.3.2 Testing the homography-based tracker

Numerical tests

In order for the correct meta-target associations to be created based on the plane-induced homography H_π, the scaled symmetric transfer dis-

tance (SSTD) for a given target pair must be both less than any other pair involving either of those targets, as well as less than the threshold τ_ϵ.

The first step to testing the homography is to find a set of corresponding points on the ground plane. At least four points must be chosen; more is better. This is done by a person, not automatically by a computer. Since the cameras are not moving, the point locations will be constant for all frames in each video sequence.

Then, using the DLT, a truth homography can be found. Using this homography, the points can be checked for consistency using the SSTD – large errors indicate that the point matches are outliers. This part of the test is done to ensure that the points have been precisely and correctly identified.

Once a set of corresponding truth points have been determined, the algorithm is run. Eventually, the ground plane-induced homography will be found, and after that time it will be constantly updated as more and more markers are added to the corresponding point list (see Section 3.2.5). The homography is recorded at certain times:

- The first time it is declared valid, *i.e.* just after enough of the markers have been declared non-collinear in both frames.
- At intervals after that time.
- At the end of the sequence.

The last homography should be the most accurate, since it is based on the most corresponding points (recall that the DLT algorithm gives a least-squares fit to the input points).

Once the homographies have been recorded, the SSTDs are calculated for each of the truth points found by the operator. To be judged a success, each homography must have scaled symmetric transfer distances less than the threshold τ_ϵ. This means that if the corresponding points were target feet features, then they would be associated into meta-targets. Assuming that it meets that standard, the best homography will have SSTDs of zero, or at most on the same order as that of the truth homography.

For a mono-numeric comparison, the mean of the scaled symmetric transfer distances can be used to compare homographies.

Visual test

By using standard image manipulation techniques, the image of one camera's ground plane can be transformed and re-sampled into the coordinate system of the other camera. The two images can be overlaid using alpha-blending. If the homography is perfect then the two ground planes will overlap perfectly, and corresponding points will lie on top of each other. Errors in the homography will show up as a disparity in the locations of features on the ground plane.

This method, a simple visual comparison, should be a quick way to determine whether a homography accurately represents the geometry of the scene.

Note that the visual comparison can only be made for points on the ground plane π. Points off of the ground plane are not represented by the two projective transformations that make up the homography found by this algorithm (those two transformations are $H_{A\to\pi}$ and $H_{\pi\to B}$). Therefore, points off of the ground plane will not overlap in the melded image.

Artificial occlusion testing

If a sequences does not include natural occlusions, we can create artificial occlusions by interrupting the program and deleting meta-target associations. When the algorithm is re-started, it will see targets in each frame that appear completely new, as if they had magically appeared. If the targets are correctly re-associated then the algorithm works properly.

3.4 Alternative methods

The method described above has some significant advantages over many other multiple-camera tracking algorithms: it requires very little user intervention and aside from the ground plane, does not require particular in-scene content. So long as the ground-plane requirement is satisfied,

after the operator selects a pair of partially overlapping cameras there remains nothing to do but hit the "go" button and let the system produce multiple-camera tracking results. As discussed in Section 1.2, the system was designed for near-fully automatic operation and un-sophisticated operators.

However, it is possible to build trackers that have other sets of restrictions. If additional information is available then alternative tracking methods might be used, or the current tracker could be made more robust. In this section various methods of obtaining and using additional pieces of information are discussed.

3.4.1 Improving this method

How might the present algorithm be made better? Can parts of the algorithm be removed entirely if additional information is available? The answer to both questions is yes, provided we can ask the operators to perform additional duties.

Closed training

The present method for finding FOV lines relies on targets naturally crossing the lines during a training period. However, these crossings could be triggered by a "planted" target. For instance, a person with knowledge of the camera setup could enter the surveilled area and walk across the FOV

lines at various points along the line. This would guarantee that every recoverable FOV line would be correctly identified, and so could be used for meta-target association shortly after the system is activated.

Replacing FOV lines

Consider the FOV part of the algorithm, discussed in Section 3.2.3. The purpose of this system is to create an initial set of meta-target correspondences, after which markers are dropped and the homography is calculated. This system could be replaced by an operator. Given two video streams, the operator could be asked to indicate corresponding target projections. This could be done with, for example, two touch screens displaying single-camera tracked targets with a glowing aura. When the operator touches one target in each camera at the same time, the system acknowledges the input, perhaps by changing the colour of the targets to be the same, and then lets that new meta-target start dropping markers.

Even more simply, the operator could be asked to directly draw the FOV lines as seen by camera A. This would jump-start the meta-target creation process, since the system would not need to wait for FOV events before starting to create associations.

Directly specifying the homography

The system of dropping markers could be rendered irrelevant by an operator. The system essentially creates a list of corresponding point pairs. An operator could create this list fairly easily with any number of tie-point selection tools, such as Matlab's cpselect() tool.

Naturally, the operator would have to be instructed to only pick points on the ground plane, so as to prevent calculation of an incorrect homography. This is almost exactly the method used in [23].

This method could be undermined by a featureless ground plane. If less than four non-collinear corresponding point pairs are visible then the homography will not be findable. In this case the operator could use the truth-point selection tool used in this thesis, marking only points on a target that are clearly on the ground plane such as one of the target's feet or the estimated location of the feet feature.

After the initial homography was specified by the operator and used by the multiple-camera tracking algorithm, the marker system could be activated. This would increase the number of corresponding points and should therefore increase the accuracy of the homography.

At this point it should be noted that the foregoing methods all require accurate operator input. This is significant, since this sort of information might not always be available. The system discussed in this thesis does not need this sort of input.

3.4.2 The fundamental matrix

The fundamental matrix F links the images of a 3D world point in two cameras. Any point in camera A leads to an epipolar line $\ell_B^T = F\mathbf{x}_A$ in camera B. The projection of the world point in camera B, $\mathbf{x}_B$, will lie on the epipolar line and so $\mathbf{x}_B\ell_B^T = \mathbf{x}_B F\mathbf{x}_A = 0$. This geometry was previously shown in Figure 2.1 on page 18.

Although the fundamental matrix is a 3×3 matrix, it is singular, of rank 2, and only has 7 degrees of freedom. The right orthonormal basis of the null space of F is the camera A epipole $\mathbf{e}_A$, the pixel location in camera B that is the image of camera A. The orthonormal basis of the null space of F^T is the transpose of the camera B epipole $\mathbf{e}_B$, the image of camera B as seen by camera A.

The fundamental matrix therefore links two cameras, but it is *not* restricted to linking points on a ground plane like the ground plane-induced homography used in this thesis. If the fundamental matrix is known, it should be possible to create an algorithm better than the one discussed in Section 3.2.7. Rather than the SSTD measure, the new algorithm might use a distance measure based on each point's distance to the epipolar line.

To track objects in an airport apron environment, [20] uses an epipolar tracking system that was developed in [22].

Calculation of the fundamental matrix can be done in a variety of

ways:

- A corner-finding algorithm could be applied to images from two cameras. Using a robust method such as RANSAC or the Least Median of Squares (LMedS), the fundamental matrix can be calculated using a number of minimal samples from the list of corners. Corners on moving targets or on non-overlapping regions *should* be rejected by a robust method.

- If an operator is available, they could be asked to select corresponding points between two cameras. The simplest method of calculating the fundamental matrix is the normalized eight point algorithm introduced in [35]. It requires at least eight corresponding point pairs; more than eight points will give a least-squares solution for F, similar to the DLT algorithm discussed above in Section 3.2.6.

Once the fundamental matrix is found a 3D model of the scene can be calculated using basic triangulation methods for known correspondence points. A fundamental-matrix based tracker might allow corresponding points to be found using appearance-based methods, provided that the cameras were of the same modality. This includes both feet points – which could be used to create a topographic map of the ground – and other in-scene points such as corners on structures not on the ground plane. These 3D points could then be displayed with image-based tex-

tures from arbitrary viewpoints. Once a basic model is created using, say, the background images, the live targets could be injected into the "world". These targets could then be watched from an arbitrary viewpoint by surveillance operators using game-like interface controls or full-immersion stereo 3D interfaces. It would even be possible to view the model with a binocular interface, since the retrieved world model would have full 3D information for major points in the scene: visual corners and moving targets.

There are clear advantages to using the fundamental matrix. Aside from 3D world construction, targets could be tracked when a common ground plane is not present. However, using the fundamental matrix has downsides. Depending on the algorithm used, high-quality operator input may be required to set up the system. If a more automatic method is used to find the fundamental matrix (*e.g.* a RANSAC or LMedS-based method), a significant number of feature points must be both detected and shared between the two views. Also, the fundamental matrix does not directly match points to points, it matches points to lines. This may lead to problems in crowded scenes.

Calibrated cameras

A camera projects 3D world points onto points on a 2D image plane according to $\mathbf{x} = P\mathbf{X}$. Camera calibration is the act of finding the 3×4

camera projection matrix P. For a basic finite projective camera, $P = KR\left[I \mid -\tilde{\mathbf{C}}\right]$. In this equation, $\tilde{\mathbf{C}}$ is the location of the camera in the world coordinate frame, R is the rotation matrix to correctly point the camera in the world, and K is a matrix

$$K = \begin{bmatrix} \alpha_x & s & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix} \tag{3.14}$$

where the two α parameters are the scale parameters in each direction (these are usually related to the focal length), s is the skew factor, and x_0 and y_0 are the coordinate of the principal point.

Cameras can be calibrated using many methods. Some of the methods are covered in [34]. Calibration methods include using automatically-recognizable in-scene calibration targets, or manually selecting the image locations of surveyed points.

Depending on how the cameras are calibrated, a variety of methods may be used to determine a fundamental matrix or a plane-induced homography. For instance, for two general cameras P and P', $F = [\mathbf{e}']_\times P' P^+$, where P^+ is the psuedo-inverse of P and $[\mathbf{e}']_\times$ is the skew-symmetric form of the epipole of the first camera (so $\mathbf{e}' = P'\mathbf{C}$ with $P\mathbf{C} = 0$) [34].

More cameras

The algorithm developed for this thesis uses pairs of cameras. Also, it only looks for FOV events in one camera of the pair. The algorithm could be enhanced to operate both ways, looking for FOV events in both cameras simultaneously in an effort to find meta-target correspondences quicker.

What happens if a third camera (camera C) is added to a currently-existing pair of cameras (A and B)? In this case camera C could be paired with overlapping camera B, and the inter-camera homography H_{BC} could be found with the usual method. At this stage two homographies are known: H_{AB} and H_{BC}. However, since the cameras all share a common ground plane, the missing homography could be easily calculated with $H_{AC} = H_{AB}H_{BC}$.

If cameras A and C do not actually overlap then the resulting homography will not project feet points to locations in the frame of camera C; this can be detected and the A-C pairing automatically rejected. However, if cameras A and C do overlap then the multiple-camera tracker could use H_{AC} for tracking.

Many of the advanced chapters in [34] are devoted to three-view and n-view geometry. The trifocal tensor and other geometric constructs can be used similarly to the fundamental matrix, and are recoverable using similar methods to those discussed above. In general, using these relationships requires finding a number of corresponding points between

the cameras, calculating a geometric relationship (possibly using a robust method), and then using that relationship to project where a target feature will show up in another camera. The projected location can be compared to the actual locations of the targets in the frame using a distance metric of some sort, and meta-target relationships created. The method described in this thesis is an implementation of this generalized algorithm: the relationship is the plane-induced homography between two cameras and the distance metric is the scaled symmetric transfer distance.

In this thesis a priority was placed on automatic functioning with minimal operator input. As described above, other methods could be implemented that use any amount of operator input. The common downside to gathering more input is that the person installing or operating the system needs to periodically spend time performing activities that do not directly relate to tracking targets. The upside to the present algorithm is that this is not required – the operator can concentrate on watching targets instead of operating the surveillance system.

Chapter 4

Implementation details

This chapter explains exactly how the various algorithms presented in Chapter 3 were implemented. The goal of this chapter is to give the reader details on how to create a working computer program.

Reading Section 4.1 is recommended, since that section covers the Generator program, written to simulate multiple-cameras in a simple world. The output of the Generator is used to test the algorithm in Chapter 5.

The remainder of the chapter can be safely skipped by those not looking for implementation details. The sections following Section 4.1 cover:

- Threshold selection. Since many of the thresholds were set using a heuristic process, typical values are given with explanations of how those values were determined.

- Specific tests. Some algorithms require specific tests to be passed before they should be used. Those tests are outlined and where necessary, their implementation explained.

The Generator program was written in C++. All other code was written in Matlab using student version 2007a. All code should be available from the same source as this document. If not, email the author at m@tthew.ca.

4.1 The Generator

As the multiple-camera tracking software was being developed, a need for test data was identified. Early attempts to use real-world video sequences added potential sources of error into the system. It was unclear whether any given error was caused by errors in one of the the multiple-camera tracking functions, or in the background subtraction or single-camera tracking code.

To satisfy this need, a program was written using C++ and the OpenGL graphics library to simulate people moving around a small world, seen from two vantage points. The program is known as the Generator. Some of the features of the Generator are:

Independent extrinsic camera parameters The world location and orientation of the two cameras are independent of each other. Both are

set in world coordinate terms directly in the code. These parameters were set with a call to gluLookAt().

Independent intrinsic camera parameters Similarly, the intrinsic properties of the cameras are independent. As a result, it was possible to set the cameras to different resolutions (*i.e.* frame sizes) and focal lengths. Although it is possible to do so using hand-coded matrices, this is complicated. Instead, the parameters were set with calls to glPerspective().

Random target behaviours The people walking through the scene appear at semi-random locations. They initially are set to walk towards a goal area around the centre of the world coordinate system, which is approximately where the cameras are pointed. However, each target's velocity vector is randomly disturbed by a small amount each frame. This means that targets do not walk in boring, straight lines, but instead they speed up and slow down, changing direction in un-predictable ways.

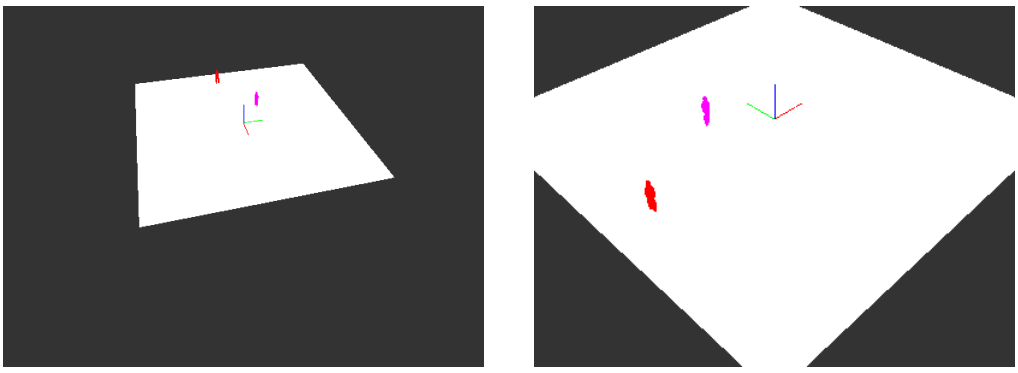
Simple target appearance - shape Targets are rigid. Each target is made up of four ellipsoids: a head, a torso, and two legs. All targets are the same size. The orientation of the legs in the world coordinate system is dependent on the target's velocity. Targets appear to move in the direction indicated by their legs, not a direction that would

require side-stepping or un-naturally angled walking.

Simple target appearance - colour Targets are red, green, blue, cyan, yellow, or magenta. There is only one light in the scene, and it is white. No significant lighting effects are used. Because OpenGL is a raster-based graphics library, no shadows exist. As a result, in the output images the targets have colour values of 255 in the appropriate channels. This makes the targets trivial to segment. Furthermore, it makes background subtraction easy. This leads to high-quality single-camera tracking results. Having high-quality input to the multiple-camera tracking code reduces the amount of code that must be searched when looking for bugs.

Image output The Generator uses the ImageMagick magick++ library to write frames to disk. The output of the program is two folders containing thousands of numbered Portable Network Graphics (PNG) images. Using images enabled the use of MATLAB's imread() function.

Synchronized output Frames from different cameras with the same number area taken at the same instant. There is no target motion between writing one image to disk and writing the other camera's image to disk. As a result, it is guaranteed that the targets will be in geometrically-consistent positions from frame to frame.



(a) Camera A – resolution 600×450 (b) Camera B – resolution 400×300

Figure 4.1: A typical scene from the Generator. From the point of view of camera A, camera B is located somewhere in the top-left of the frame.

Alignment cues

Figure 4.1 shows a typical pair of frames from the Generator. The world is structured with the following characteristics:

- The red, green, and blue lines near the centre of the frames indicate the world X, Y, and Z axes, respectively. Each line is one unit long. The lines join at the world origin.
- The white square is on the $Z = 0$ plane. It covers $X = [-5 \ldots 5]$ and $Y = [-5 \ldots 5]$.
- Targets can move in the range $X = [-10 \ldots 10]$ and $Y = [-10 \ldots 10]$. If they exceed this range then they fall off of the edge of the world (the world is flat), and so are no longer displayed.

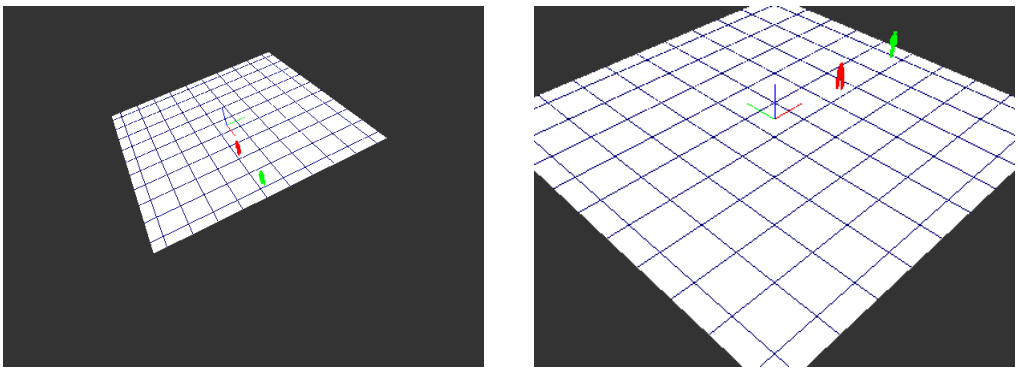
90% of the targets start at a random position on the edges of a square with boundaries at $X = \pm 8$ and $Y = \pm 8$. The other 10% start in random locations inside that square.

Each target's initial velocity vector points to a random location inside a square with boundaries at $X = \pm 4$ and $Y = \pm 4$. This means that they start out walking towards somewhere near the centre of the world. This helps to improve the chances that a target will trigger FOV events in camera B.

The number of targets in the world at any given time can be limited. Limiting the number of active targets limits the number of distractor points in the maps used to find the FOV lines. The maximum number of targets was typically set to three.

Section 3.3.2 called for a number of corresponding test points to be found on the ground plane. Because the Generator world is somewhat barren, a grid was overlaid on the white square. The resulting world has many intersections on the ground plane that can be easily picked out by hand. Figure 4.2 shows the gridded plane from Run 17, where camera A was tilted by about twenty degrees.

In this thesis three runs are discussed. The runs are numbered 17, 18, and 23. The numbers are based on the dates they were created, so, for instance, there is no run 19 because the Generator was not run on the 19^{th}.



(a) Camera A (b) Camera B

Figure 4.2: To be able to find corresponding points, a grid was overlaid on the world ground plane. The grid has unit spacing, and is aligned on the half units of the world X and Y axes.

4.2 Background subtraction

Of all the algorithms implemented in support of multiple-camera tracking, the background subtraction algorithm contains the largest number of critical threshold values. These values are critical because if they are not set properly then targets will not be detected, shadows will be included in target masks, parts of targets will be called shadow, and background pixels can be flagged as targets. The thresholds in this algorithm are:

- τ_H, the hue difference threshold for shadow detection
- τ_S, the saturation difference threshold for shadow detection
- α and β, the range of value ratios for shadow detection

- T_{low}, a low threshold to determine which pixels are sufficiently different from the background to warrant further inspection
- T_{high}, each blob must contain a pixel that exceeds this threshold to be considered for further inspection
- T_{Area}, the minimum size, in square pixels, of an MVO blob
- T_{AOF}, the minimum average optical flow of a target blob

No automatic methods to select the thresholds were presented in any of [28,29,30]. In [29] a preliminary sensitivity analysis was carried out on the shadow detection thresholds. Ten combinations of the four variables were tested.

Further qualitative testing of the shadow detection module was performed through visual inspection of the various sub-results. Unfortunately, we were unable to come up with any specific recommendations. The ten combinations of values tested in [29] do not appear to have been chosen with any sort of plan, and do not lead to particularly good results

on the test sequences. Currently, the following settings are used:

$$\begin{aligned} \tau_H &= 0.4 \\ \tau_S &= [0.08 \ldots 0.3] \\ \alpha &= 0.2 \\ \beta &= 0.8 \end{aligned} \tag{4.1}$$

Note the range given for τ_S. This parameter needs to be tuned depending on the strength of the shadows.

The remaining parameters govern how to determine which pixels are part of an interesting blob, and which blobs are actually part of the background. We discuss each parameter in turn.

No guidance is given in any of the original papers on how to set T_{low} or T_{high}. T_{low} was set to 0.15 times the maximum value found in DB (originally defined in Equation 3.2 on page 25). Similarly, T_{high} was set to 0.42 times the maximum value. These values were found by a qualitative visual inspection of the results, seeking to get enough of the target identified in the blob without making the blob too large.

The area threshold T_{Area} was set to a fixed number of pixels in the original papers. However, in this research we dealt with varying frame sizes. As a result, a fixed threshold would not work. We therefore set the

area threshold to a multiple of the area of the frame:

$$T_{Area} = 10^{-3} \times (width \times height) \tag{4.2}$$

Because the targets in the Generator were much smaller than targets in the real-world scenes, the threshold was reduced by a factor of 10 when processing simulated scenes.

The average optical flow threshold T_{AOF} was set to a fixed number in the original papers. Again, due to the different frame sizes, this method would not work here. Because optical flow is measured in units of pixels, we set it to a multiple of the diagonal size of the frame.

$$T_{AOF} = 0.8 \times 10^{-3} \sqrt{width^2 + height^2} \tag{4.3}$$

Other background subtraction notes

The algorithm takes the median of a number of recent images to update the background model. The suggested number of images from [30] is $n = 7$, with 10-frame spacing and a single duplicate of the current frame. We found that $n = 9$ with 7-frame spacing and a single duplicate of the current frame worked well.

When profiling the background subtraction code, it was found that calls to MATLAB's `median()` function were taking up a 50% of the run time. The next longest function was `rgb2hsv()`, which transforms RGB images

into the hue, saturation, and value (HSV) colour space. Both of these functions could be re-written to be significantly faster than their MATLAB implementations, but at the cost of reduced accessibility and flexibility.

4.3 Single-camera tracking

Of all the functions that make up the single-camera tracking algorithm described in [5], none of them remained un-modified during this research. At the beginning of the research the system worked correctly on the test sequences used in [5], each less than 200 frames long and each designed to show a particular case (*e.g.* occlusion, crossing targets). However, the algorithm did not work on any of the longer test sequences used for this research, including the movies available in the PETS database [36]. Failures were typified by "hard" crashes such as reading beyond the end of a matrix, rather than qualitative errors or "soft" crashes stemming from faults inherent in the logic of the algorithm.

A number of changes were made in order to fix the single-camera tracking algorithm. In general, the overall logical structure and algorithmic flow of the algorithm was kept the same. The changes made to the algorithm fall into three main categories:

1. Replacement of the background subtraction components,

2. Improvements or creation of functionality, including bug fixes, and

3. Improvements in speed.

Background subtraction

The original algorithm used a static, pre-computed background image to detect foreground targets. This led to significant problems with scenes longer than a few hundred frames – eventually large chunks of the frame would be identified as foreground. Furthermore, objects that exhibited slow motion from frame $t-1$ to frame t were discarded because the algorithm's inter-frame motion detection was naïve.

In this research, the background subtraction part of the single-camera tracking algorithm was replaced with the algorithm described in Section 3.2.1, whose implementation details are found above in Section 4.2. This significantly improved the detection of foreground objects. With the previous algorithm, targets would often disappear for one frame if their motion was insufficiently salient, but then re-appear in the next frame. With the current algorithm this problem disappeared.

Functionality fixes

As previously mentioned, the single-camera tracker was effectively non-functional due to frequent crashes. Many fixes were made, too numerous to be fully described here. Some of the more important fixes include:

- A monotonically-increasing target counter. A new target now re-

ceives a unique label instead of receiving a previously-used label.

- The shape matching code was originally written to fit segments with a B-spline using 60 control points. The number of control points was a fixed constant. Some segments targets in the long test sequences had perimeter lengths shorter than 60 pixels. As a result, the shape matching code failed – it is impossible to have more unique B-spline control points than there are pixels in the perimeter of an object. In the current code, the number of control points was set to be 3/4 of the circumference of a circle with the same area as the segment. Since a circle has the minimum circumference for a given area, this guarantees that the number of B-spline control points will be less than the number of perimeter pixels.

- Many morphological operations were changed to be dependent on the frame size. The sizes of the various structure elements used to enlarge targets and detect overlaps were made dependent on the diagonal size of the frame instead of being hard-coded.

- At one point the algorithm needs to find points of high curvature on the boundary of an object. The existing algorithm did not do this effectively, and gave incorrect results. That algorithm was re-written to provide correct results. As a happy side effect, the new code was found to be much faster.

Speed fixes

Many functions or individual stanzas were re-written to increase execution speed. The final speed increase was on the order of 50× – from one frame every thirty seconds to about two frames per second. This speed increase is especially evident during object merges, when the shape matching code is called.

Some of the speed increases were possible by re-writing algorithms to use matrix operations. In MATLAB matrix operations are almost always significantly faster than numerically equivalent operations using for() loops and the similar structures. This is a result of history: MATLAB was originally built as a front-end to the EISPACK and LINPACK linear algebra libraries. Other improvements were made by changing a function's logic to reduce or eliminate the number of calls to expensive functions such as circshift().

4.4 Finding FOV lines

Four significant changes were made to the algorithm discussed in Section 3.2.3, originally found in [24].

First, as in [25], an FOV event was defined to occur when a target completely entered the frame, or, when exiting, at the first instant that it begins to touch an edge of the frame. This has the effect of moving the

FOV lines inwards from their actual locations. However, it also increases the accuracy of the lines, since partially-visible objects are not used to find the lines. Although it is possible to run the feet feature finding algorithm on a partially-visible target's mask, it is far from clear that the output will be the location of the feet of the target. Indeed it is possible that the target's feet are not even in the frame, although the feature finder will give an output regardless. Making this change ensures that the feet are visible, and so the feet feature finder has a good chance of finding the correct point.

The other three changes have to do with the maps of points used to calculate each FOV line. When an FOV event has been triggered from camera B, the original algorithm adds the feet locations of all targets in camera A to a map. Eventually, the algorithm uses a Hough transform to find the FOV line. However, in some sequences the background subtraction algorithm created foreground targets that lasted for a very small time – usually one frame. The false targets were in the areas such as tree-lines on the horizon, where high-frequency wind-driven leaf motion appeared as target motion. These targets were correctly given labels by the single-camera tracker, but disappeared in the next frame. Naively including the "feet" of these spurious targets effectively added distraction points to the map of feet locations, potentially confusing the Hough transform.

To combat these spurious targets, a minimum age requirement was

added. In order to be included in the target map, a target must have been visible for at least one frame before the current frame. This simple requirement prevented inclusion of many of the distracting tree-line and cloud targets.

The third change was to increase the minimum number of FOV events before the FOV line is computed. In theory, two FOV events should be enough to define a line. However, if multiple targets are present in camera A then their feet will distract the Hough transform, and the resulting line will not necessarily be correct. No guidance is given in [24] on how many FOV events should be counted before computation of the FOV line. We raised the minimum number to six events. Changing this threshold effectively changes the training period before meta-target associations can be created. A higher number means more training is required, but the algorithm will be more resistant to high-traffic scenes with multiple distracting targets. If we had had scenes with more distracting targets (*i.e.* higher-traffic scenes), then the minimum number of FOV events could be raised even higher.

The final change has to do with the actual method used to compute the FOV line. The original paper uses a Hough transform on the list of points. Instead of this method, we decided to use the Random Sample Consensus (RANSAC) method. Originally introduced in [37], RANSAC uses minimal samples of the data to fit a model. Points are then classified into

inliers and outliers depending on how well they fit the model. The model from the minimal sample that yields the maximum number of inliers is the winner – it fits the data the best.

In this case the model is the slope and intercept of the line, and the minimal set needed to find these parameters is two data points. We set the inlier threshold at 1% of the diagonal size of the frame. The original RANSAC implementation was found at [38], although we speeded up the code by replacing some stanzas with matrix operations. At the end of the RANSAC algorithm the final output FOV line is calculated by taking a least-squares fit of the inliers.

In this implementation, the RANSAC method took approximately 2-4 times longer to fit the same line. When more markers were used, RANSAC slowed down. However, the actual time taken was very small – fitting 1000 points with RANSAC takes about 0.01 seconds on the machine used for this research (a 1.83GHz Apple Macbook), and is far from the slowest step in the system.

4.5 Dropping markers

There are two thresholds and a timer used to determine whether a marker should be dropped in any given frame.

1. In order to drop a marker, both targets must be fairly close to the

median of their heights in the past few frames. This prevents dropping a marker just as a target enters occlusion. The number of historical frames to be considered was set to 10, and the threshold to $\pm 25\%$.

2. Markers should not be dropped too close to the previous marker. Therefore, at least one of the targets in the pair has to move more than a minimal distance. This distance was set to 3% of the diagonal frame size.

3. Assuming that the previous conditions were satisfied, there is no reason why we should not drop a marker every frame. In the function there is a timer that can be set to ensure that markers are only dropped after at least n frames have passed since the previous drop. After experimentation, it was found that there was not much benefit from reducing the number of markers in this manner, so the timer was set to 1 – so long as the height and distance conditions are met, a marker will be dropped every frame.

4.6 Calculation of a homography

If either of the sets of points ($\mathbf{x}_A$ and $\mathbf{x}_B$) used to calculate a homography is collinear, then the homography will be degenerate. Such a condition might occur near the start of dropping markers. If one target walks in

an approximately straight line, they might drop four or more markers (enough to calculate a homography), but the points will be degenerate for the purpose of predicting target location off of the line.

To detect this condition, the RANSAC line-fitting function that was used to find FOV lines was re-purposed. After fitting a line to the markers, the number of inliers as a fraction of the number of input points is calculated. There are two thresholds at work here:

1. The inlier threshold, measured in pixels, determines whether a given point is close enough to the model line to be called an inlier. This threshold was set to 2.5% of the diagonal frame size for all sequences except the arena sequence, where it was set to 0.8%. Reducing this threshold means that more points will be classified as outliers. More outliers means that the data is more likely to be called non-colliner. Therefore, lowering the threshold means that we are more likely to call the data non-collinear, and are therefore more likely to say that the homography is valid. Lowering the threshold means nearly-collinear data is more likely to be declared valid.

2. The number of inliers as a percentage of the total number of points is set to 90%. This means that at most 90% of the points can be inliers – if there are more inliers than this then the data is called collinear and the homography is not calculated. Raising the threshold above

90% means that fewer outliers (*i.e.* non-collinear points) are required before the homography is calculated and declared valid.

4.7 Homography-based multi-camera tracking

4.7.1 Thresholds

There are two thresholds in this function. The first, τ_ϵ, determines the maximum symmetric transfer distance for a pair of targets to be associated. The other is the change reticence threshold, which determines how much better a pair has to be to replace a pre-existing meta-target.

Recall the first part of Equation 3.10, which defines the scaled symmetric transfer error ϵ:

$$\epsilon = \frac{\|H_\pi \mathbf{x}_A - \mathbf{x}_B\|}{d_B} + \frac{\|H_\pi^{-1} \mathbf{x}_B - \mathbf{x}_A\|}{d_A}$$

The threshold was set to $\tau_\epsilon = 0.2\sqrt{width^2 + height^2}$, *i.e.* 20% of the diagonal frame size. Setting a higher threshold means that matches could be created between targets that are farther apart. This means that bad matches might accidentally get created.

The second threshold is τ_Δ, the change reticence threshold. This is multiplied with the SSTD of the best of the previously-created meta-targets

threatened by the prospective pair of targets. If the distance of the prospective pair is lower than the reticence distance then the old meta-target(s) are deleted and a new meta-target association is created. This threshold was set to $\tau_\Delta = 0.75$.

4.7.2 Speed

The multiple-camera tracking function was found to be fairly fast compared to the single-camera tracker and the background subtraction algorithm. The actual tracking is a very quick process, with most of the time being taken by the feet feature finder (which is itself speedy). When many targets are in the scene, some form of caching locations of the feet feature and removing redundant calls to the function would speed up the system, but was not implemented in order to keep the code simple.

When the list of markers becomes large ($> 2,500$) the DLT algorithm, and specifically the SVD contained therein, starts to become the slowest part of the multi-camera tracker. However, even when the DLT is slow, it is still an order of magnitude faster than the single-camera tracker. An experimental change was made after most testing was completed to sample the marker list so that only the last 1,000 markers were used. This did ensure that the time taken was both consistent and small. The effect of this change on the whole system was not tested. All results reported in this thesis were made using all of the markers available.

Chapter 5

Results and discussion

5.1 Feet feature finder

Recall that the feet feature is the single point that represents the target's location on the ground plane. It is represented by a pixel location in the image's coordinate system.

5.1.1 Comparing to hand-found points

The data in Table 5.1 was found by calculating the distance between the manually identified feet feature and the point identified by each algorithm. Two algorithms were tested. The first was the widely-used method of finding the centre of the bottom edge of the bounding box. The second is the method described in Section 3.2.4, which rotates the target mask

upright, then finds the point between the two edge points closest to the bottom corners of the bounding box.

The raw mean distance between the point pairs was measured. It was then divided by the diagonal frame size for a resolution-independent measurement. The ratio between the bounding box and rotated measurements was taken: $ratio = d_{rot}/d_{BB}$. Ratios greater than 1 indicate that the bounding box method performed worse, on average, than the rotated method.

The data shows that in all sequences the rotated method described in Section 3.2.4 performs better than the widely used bounding box method. In the un-tilted real-world sequences the bounding box method produces results 28% more distant from the hand-identified location than the rotated method. In the two sequences where the camera was tilted about twenty degrees, arena A and Generator 17A, the rotated method performs much better than the bounding box method.

However, despite the improvement, it should be noted that the difference between the two methods in units of pixels is actually fairly small.

5.1.2 Comparing meta-target creation distances

Knowing how well the feet finders perform with respect to a human observer is good, but how well do the feet feature finding algorithms perform in practice? The scaled symmetric transfer distance, ϵ, was recorded

Table 5.1: Data on the performance of the bounding box (B.B.) and rotated feet feature finders, from both Generator and real-world sequences.

Sequence	Diag. frame size	Number of test points	Algorithm	Raw mean dist (pix)	Scaled mean dist $\times 10^3$	Ratio
Gen. 23A	750	203	B.B.	2.66	3.55	1.97
			Rotated	1.35	1.80	
Gen. 23B	500	106	B.B.	3.40	6.80	1.60
			Rotated	2.12	4.24	
Gen. 17A (tilted)	750	55	B.B.	3.28	4.37	2.56
			Rotated	1.28	1.71	
Gen. 17B	500	101	B.B.	2.77	5.54	2.16
			Rotated	1.28	2.56	
Gym A	433	108	B.B.	4.60	10.6	1.39
			Rotated	3.30	7.63	
Gym B	433	108	B.B.	5.93	13.7	1.27
			Rotated	4.68	10.8	
Arena A (titled)	433	56	B.B.	5.48	12.7	2.85
			Rotated	1.92	4.44	
Arena B	433	42	B.B.	6.00	13.9	1.37
			Rotated	4.39	10.2	
Field A	433	101	B.B.	4.70	10.9	1.12
			Rotated	4.20	9.72	

every time a meta-target association was created. At the same instant, the SSTD was re-computed with the feet features found using the bounding box method. The mean values $\bar{\epsilon}$ are show in Table 5.2. The actual values for each meta-target are plotted in Figures 5.1, 5.2, 5.3, 5.4, and 5.5.

Table 5.2: Mean scaled symmetric transfer distances at meta-target creation. The BB-misses column is the number of meta-targets that would not have been created had the feet been found with the bounding box method, given a threshold of $\tau_\epsilon = 0.20$.

Sequence	Number of meta-targets	$\bar{\epsilon}_{actual} \times 10^3$	$\bar{\epsilon}_{BB} \times 10^3$	Ratio $\frac{\bar{\epsilon}_{BB}}{\bar{\epsilon}_{actual}}$	BB misses
Gen. 18	143	8.40	9.70	1.15	0
Gen. 23	127	18.7	18.0	0.96	0
Gen. 17	261	27.8	29.0	1.04	1
Gym	108	66.7	90.8	1.36	5
Arena	140	47.9	90.6	1.89	7

Figures 5.1 and 5.2 show data from essentially identical sequences. The only difference from Generator run 18 to run 23 was that the people had different paths. Neither sequence had meta-targets created that would have been missed had the bounding box method been used. Interestingly, when we compare the two figures it is clear that the homography used in run 18 started out much more accurate than the homography used in run 23. This result is discussed below.

As described in Section 4.7, the threshold for meta-target creation was set to $\tau_\epsilon = 0.2$. Given that threshold, either feet feature finder would

perform quite well.

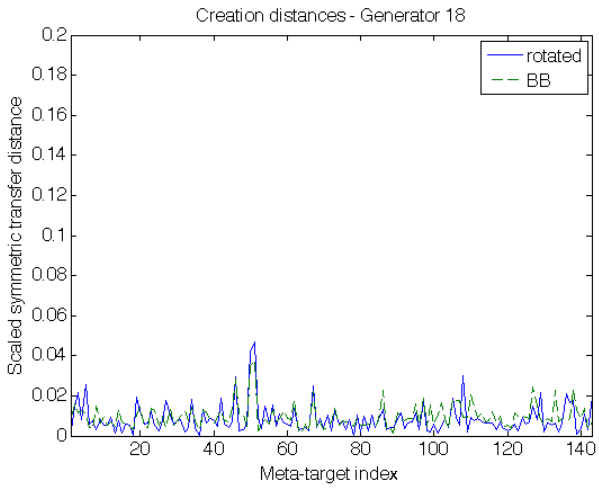


Figure 5.1: SSTD upon meta-target creation for Generator sequence 18.

The results of the un-tilted real-world gym sequence, shown in Figure 5.3 have a similar shape to that of Generator run 18. Distances for the first few meta-targets are near to the threshold. As time passes and more markers are dropped, the accuracy of the homography is increased and the SSTDs are reduced. In the gym sequence, five of the 108 meta-targets would not have been created had the bounding-box method been used to find the feet.

The tilted Generator sequence, run 17, shows similar behaviour to the gym sequence and Generator run 18. That is, the homography starts out with relatively large errors, but increases in accuracy as more markers are dropped. With one camera tilted, the bounding box distance would

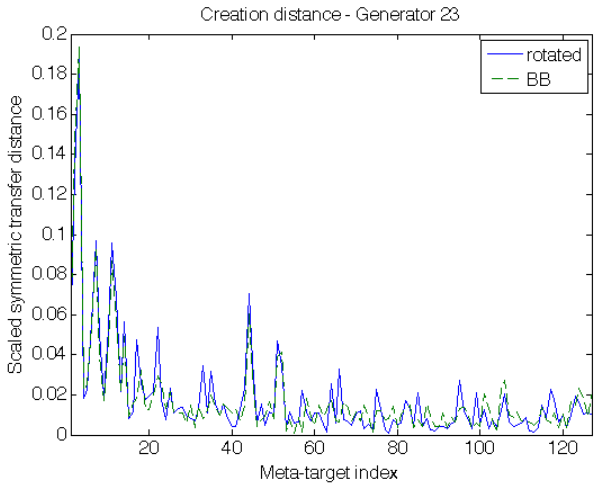


Figure 5.2: SSTD upon meta-target creation for Generator sequence 23.

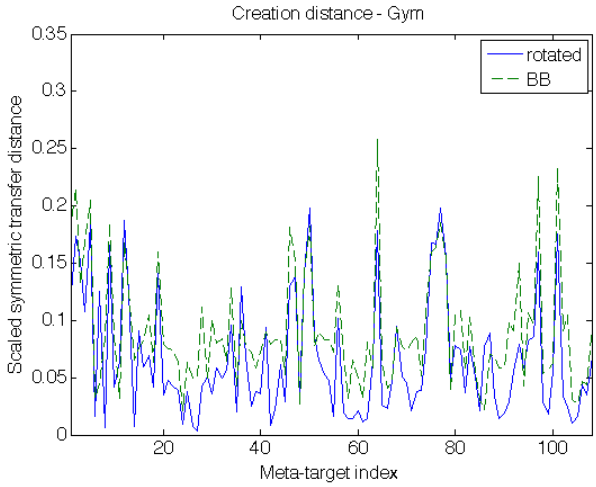


Figure 5.3: SSTD upon meta-target creation for a the real-world gym sequence. Note the five peaks where the bounding box line exceeds 0.2. Those meta-targets would not have been created if those distances were used.

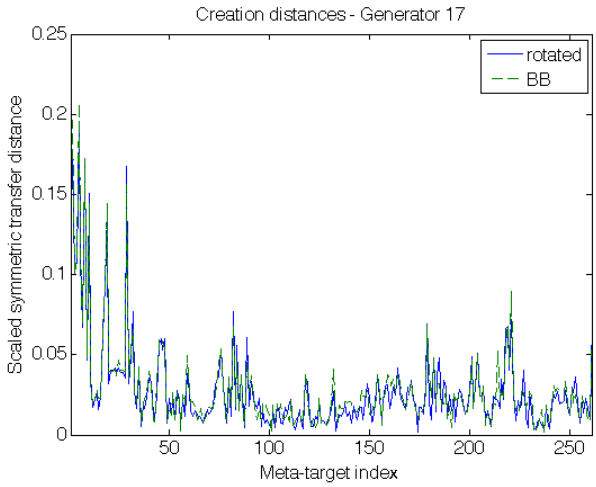


Figure 5.4: SSTD upon meta-target creation for Generator sequence 17. Camera A was tilted by about twenty degrees.

not have been below the threshold τ_ϵ for one of the meta-targets; that association would not have been created until the distance dropped below the threshold.

The arena sequence, in which camera A is tilted, clearly shows the benefit of using the rotated method to find the feet feature point. Figure 5.5 shows that even after a significant number of markers have been dropped and the homography has settled, the bounding box method still produces worse scaled symmetric transfer distances than the rotated method. Although Table 5.2 indicates that only seven meta-targets would not have been created with the bounding-box method based on a threshold of $\tau_\epsilon = 0.2$, Figure 5.5 shows that if a slightly lower threshold had been used then many of the meta-targets would not have been correctly

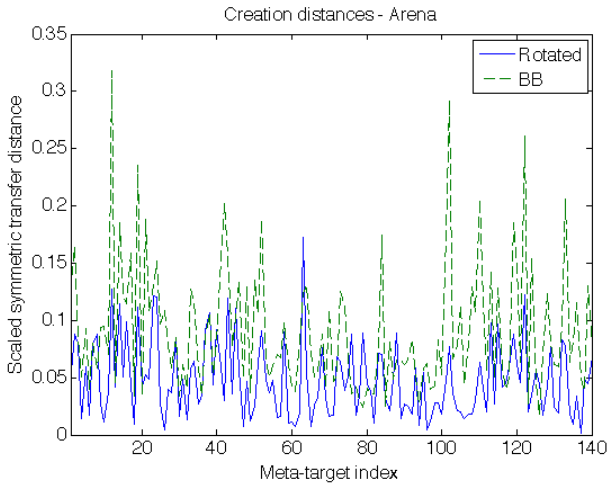


Figure 5.5: SSTD upon meta-target creation for the real-world Arena sequence. Camera A was tilted by about twenty degrees.

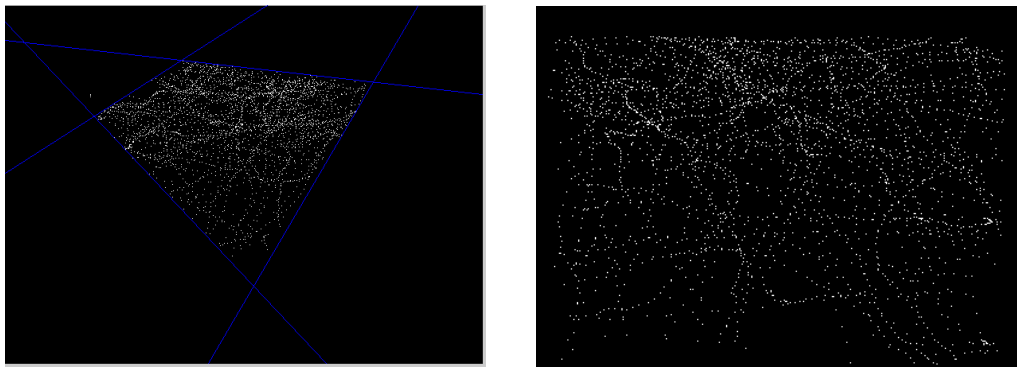
associated at all.

5.2 Homography

5.2.1 Markers

When testing the algorithm, it was often useful to examine the list of markers in graphical form. When running, this allowed an observer to watch markers being created.

Figure 5.6 shows the marker images from the Generator run 23. In addition to the marker images, the FOV lines have been overlaid on the camera A image. It is clear that the FOV lines are in the correct locations. Observe the blank space near the top of the camera B image, Figure 5.6(b).



(a) Camera A (b) Camera B

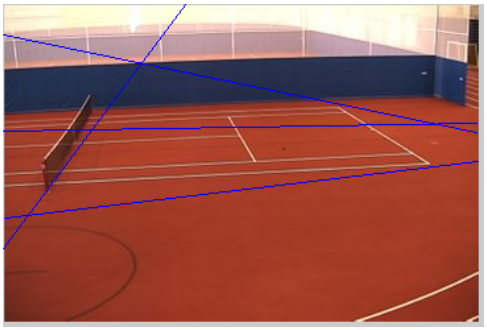
Figure 5.6: Plotting the list of markers from the Generator 23 sequence shows the extent of the overlap between the cameras. Each marker corresponds to exactly one point in the other image. The FOV lines have been overlaid in camera A.

The height of this space roughly corresponds to the apparent height of a target when it reaches the top edge of the field of view. Since markers are not dropped for targets that touch any of the FOV lines, this area does not get any markers.

Figure 5.7 shows the background model images for the gym sequence. The recovered FOV lines of camera B are superimposed on the background image from camera A in Figure 5.7(a). The final set of markers for the gym sequence, Figure 5.8, shows two interesting artifacts:

- The bottom FOV line is incorrectly placed. This is because there were very few edge events on the bottom edge of camera B, and they were all close together.

- The same border effect mentioned above for the Generator sequence



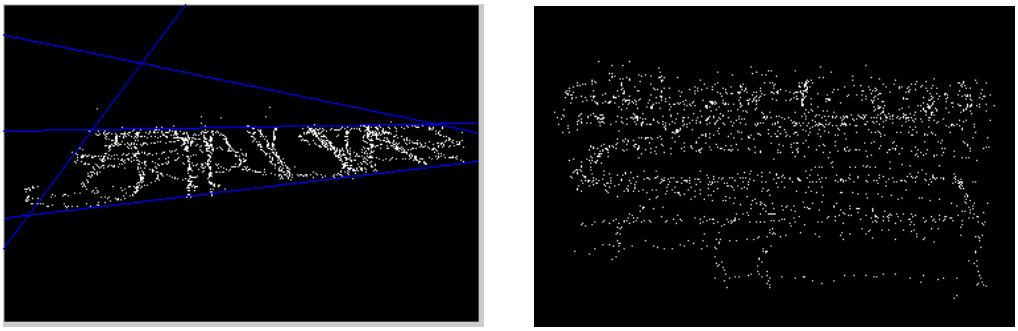
(a) Camera A, with superimposed FOV lines

(b) Camera B

Figure 5.7: The background models from the gym sequence.

is present here. In addition, there are clear borders on the left and right edges. These are all caused by the same effect – typical targets have a non-trivial height and width relative to the frame size, so by the time that they stop touching all the edges their feet are already significantly inside the frame.

Note that the neither artifact affects homography-based meta-target creation. Since meta-target associations are only created when targets are wholly inside the frame, the feet will be in the area of support of the homography before the homography will be used to calculate a scaled symmetric transfer distance.



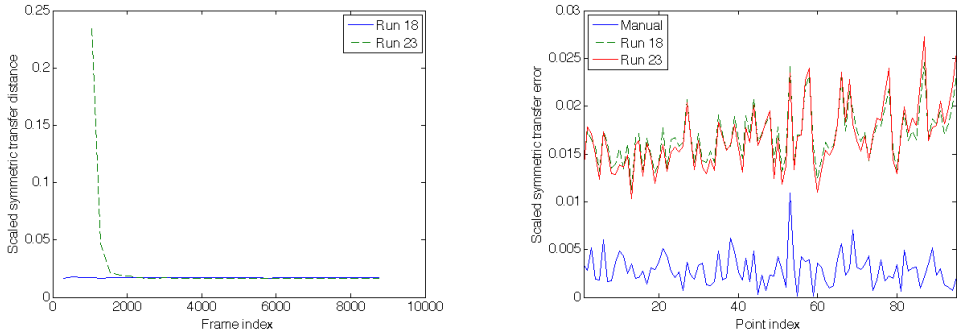
(a) Camera A (b) Camera B

Figure 5.8: These are the markers from the gym sequence. The FOV lines have been drawn the image from camera A. The bad line is from the bottom of camera B.

5.2.2 Numerical tests with truth points

For Generator runs 18 and 23, 95 corresponding point pairs were manually identified and used ground truth. During the algorithm's run the SSTD for each of those truth points was recorded at a regular interval as the homography was updated. The mean distance at each interval is shown in Figure 5.9(a). The distance for each of the 95 points using the final homography for each of the two runs is shown in Figure 5.9(b), along with the distances based on the homography that best fit the manually-identified points, *i.e.* the truth homography.

In Run 23 some of the first markers were dropped by a meta-target that was incorrectly created by the FOV line method. This resulted in relatively large SSTDS. However, the homography-based tracker quickly identified the error. As more markers were dropped by other, correctly-associated



(a) Mean SSTD of truth points (b) SSTD using final homography

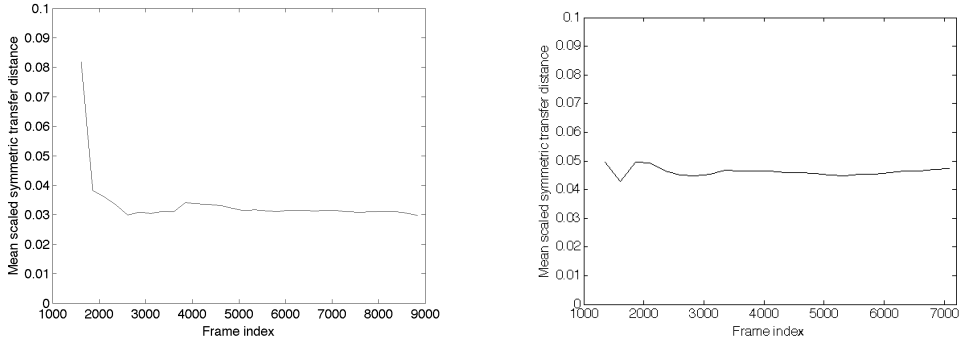
Figure 5.9: Scaled symmetric transfer distances for 95 truth points in Generator sequences 18 and 23.

meta-targets the system quickly corrected itself.

Table 5.3 shows the mean scaled symmetric transfer error for the first and final homographies, as well as for the homography created using the DLT algorithm on the manually-identified points (*i.e.* the best-fit homography). It should be noted that the final homography yields distances well below the threshold $\tau_\epsilon = 0.2$ required to create new meta-target associations.

Table 5.3: Mean scaled symmetric transfer distances for the first and final homography

Sequence	Truth points	Homography		
		First	Final	Manual
Gen. run 18	95	0.0167	0.0169	0.0029
Gen. run 23		0.2369	0.0167	
Gym	84	0.0818	0.0317	0.0124
Arena	48	0.0425	0.0416	0.0238



(a) Gym (b) Arena

Figure 5.10: The mean SSTD of the truth points for the two real-world sequences over time.

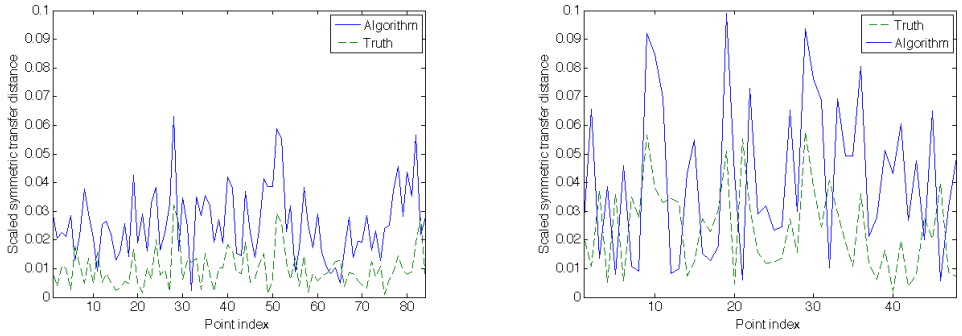
The gym sequence also started with a meta-target that was incorrectly associated by the FOV method. As more meta-targets travelled around, the homography increased in accuracy and the SSTDs levelled off. The mean SSTD values for the 84 truth points in that sequence are shown in Figure 5.11(a)

The arena sequence was fairly calm and low-traffic, so it started and finished with all-around good performance. The SSTD values of the 48 truth points used in that sequence are shown in Figure 5.11(b).

5.2.3 Visual tests

Generator sequences

Figure 5.12 shows two frames from different Generator sequences that have been melded together using standard image processing techniques.



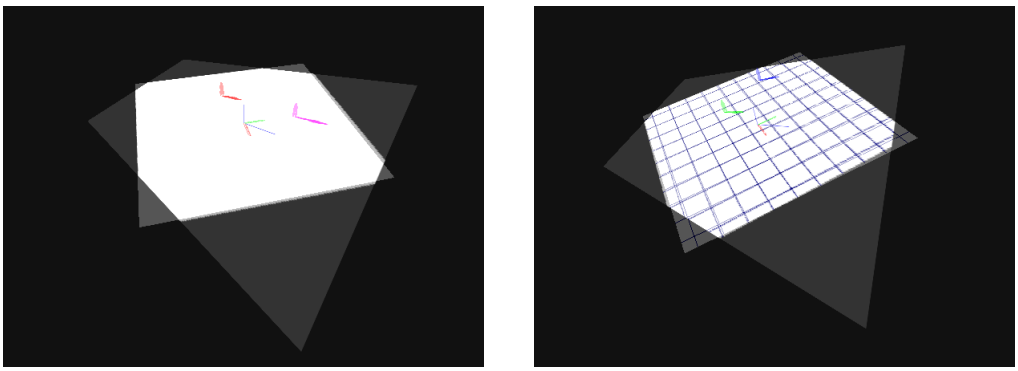
(a) Gym – 84 points (b) Arena – 48 points

Figure 5.11: SSTD of truth points using both the final algorithm and the truth homographies for the real-world sequences.

The homographies found using the multi-camera tracking algorithm were used to create the melded outputs.

In both images it is obvious that the ground-planes are not perfectly lined up. This is because the corresponding point pairs used to create the homography are not the same as points that would be matched by a human. We discuss the reason for this disparity as observed in the gym sequence starting on page 111.

Note also that the pixels of the targets in the camera B image have been projected into the camera A coordinate system as if they were on the ground plane. If the pixels in the targets' heads were thought of as being on the ground plane in camera B, then because they are above the feet pixels they will be projected to coordinates farther from the location of camera B than the feet. This manifests itself in the shadow-like appear-



(a) Generator run 23 (b) Tilted Generator run 17

Figure 5.12: Frames from two Generator sequences melded using the homography found with the multi-camera tracking algorithm.

ance of the projected targets.

Gym sequence

One way to test the validity of a homography between two planes is to transform one image using standard image processing techniques and then meld it with the other image. If features in the world line up in the melded image then the homography is valid. The two background images for the gym sequence were melded using the ground plane-induced homography found with the multiple-camera tracking algorithm. The result is shown in Figure 5.13

In order to more closely examine the results, the main section of the melded image was isolated and cropped. Figure 5.14 shows two versions of the melded image. Figure 5.14(a) was created using a homography

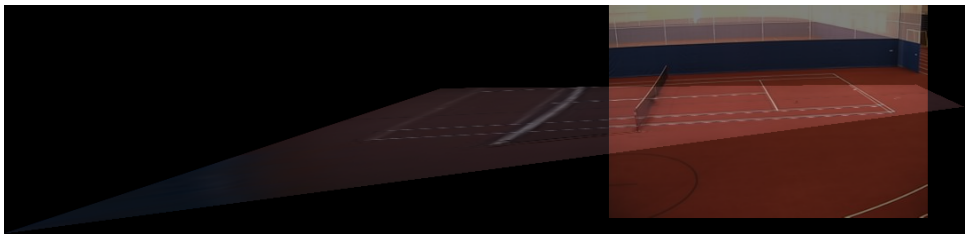
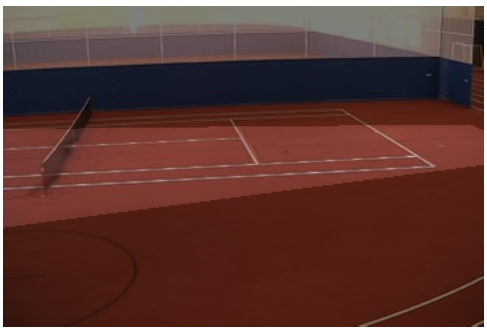
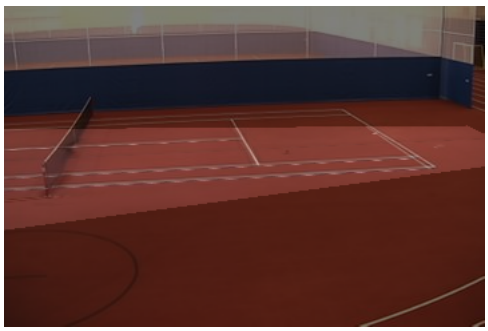


Figure 5.13: The gym background models overlaid into one image. The algorithmically-found homography was used.



(a) Manual truth homography



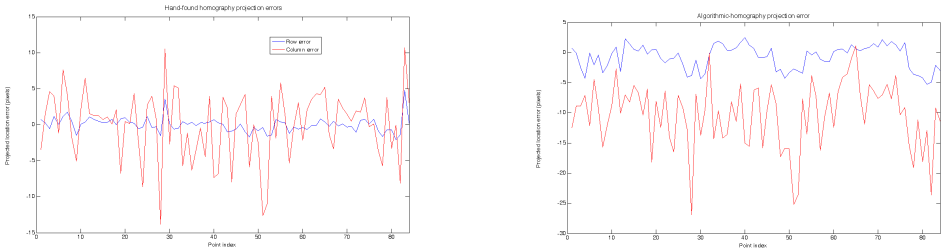
(b) Algorithm homography

Figure 5.14: A cropped section of the two melded background images, using different homographies.

found with 84 manually-identified corresponding point pairs – it is a truth image. Figure 5.14(b) was created using the algorithmically-found homography.

Figure 5.14(b) clearly shows that the algorithmically-found homography comes close, but does not line up the images correctly. Figure 5.15 shows the difference between the projected and actual truth point locations in camera B when the two different homographies are used.

In Figure 5.15(a), with the truth homography, the errors are all fairly



(a) Hand-found homography (b) Algorithm homography

Figure 5.15: Projection errors in camera B, in units of pixels, when two different homographies are used to project the points. Red is the column error, blue is the row error. Negative errors mean that the projected location is right of or below the actual location.

benign. The row is accurate, but the column is very sensitive. This is because of the relatively small range of vertical inputs in camera A – small errors in the row position of a point in camera A leads to large differences in the projected column position in camera B. However, the average error is very close to zero.

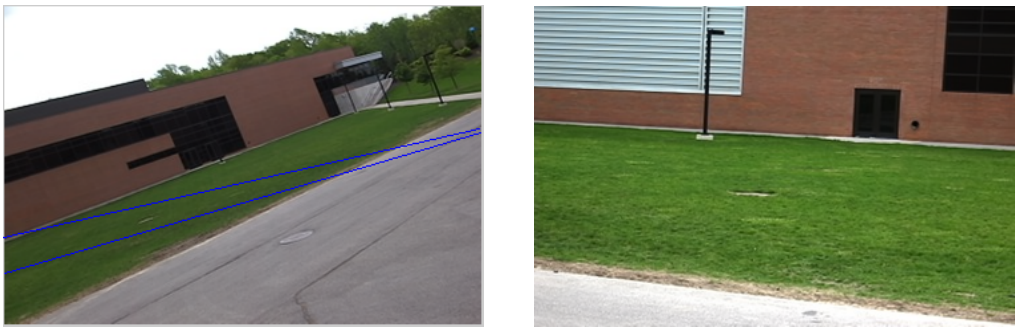
In Figure 5.15(b) the column error (red) again shows a similarly large variance, but is centred well below zero. The row error (blue) is slightly below zero. This means that points from camera A are being projected significantly to the right and slightly below where the human observer placed them in camera B. This is because of the differences in the feature points selected by the human observer and by the algorithm.

The human observer selected pairs of corresponding points with direct regard to the location on the ground plane. The size of the target mask was not seen. On the other hand, the computer only saw the mask

when it selected the feet feature location in both cameras. The feet feature became the corresponding point pair. The target mask goes through a series of morphological processes in the background subtraction and single-camera tracking algorithms. The result of these operations is that the mask is dilated when compared to the mask that a person would select. Thus, in both cameras the feet are identified to be in lower rows than a human might select.

Therefore, a human-identified point in camera A is seen by the computer to represent a target whose feet, if identified by the person, would be at a higher row. The homography found by the algorithm takes this into account, and projects that point to where the feet feature would be if it were found by the algorithm. Because we are comparing that projected location to a hand-found location, the projected feet location will be displaced to the right and slightly below the hand-found location.

The main consequence of this explanation is that plots such as that in Figure 5.15(b) can be trusted, but only so long as the geometry of the scene is taken into account when interpreting the data. This also implies that the images created by overlaying the background images should not be expected to be perfect matches.



(a) Camera A – with FOV lines (b) Camera B

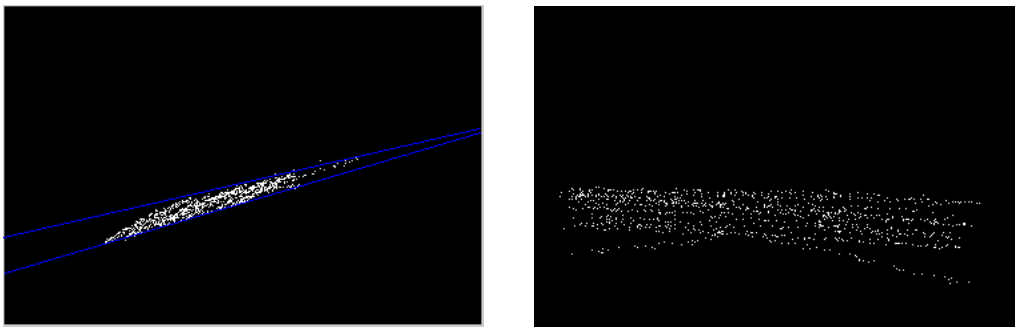
Figure 5.16: Background model images for the arena sequence with overlaid FOV lines.

Arena sequence

Figure 5.16 shows the background images used in the arena sequence. To give a sense of scale, the building in the scene is about three stories tall. The left and right FOV lines have been overlaid on the camera A image. The top and bottom FOV lines were not calculated, since no targets transited those edges in this sequence.

Figure 5.17 shows the final set of markers used to calculate the homography. Clearly, based on the two background images, camera B only covers a small slice of the total field of view of camera A. Therefore, given the large area covered by camera A, it is un-surprising that the targets did not cover much of the image. This is seen in the tight grouping of the markers.

Because the markers were tightly grouped in camera A, they were de-



(a) Camera A – with FOV lines (b) Camera B

Figure 5.17: Markers were dropped in a very small area on the image. This necessitated a lower collinearity threshold when calculating the homography.

clared collinear based on the original threshold, and the homography was not calculated. This was un-satisfactory. Therefore, for this sequence the inlier threshold used when determining if the marker data was collinear had to be reduced from from 2.5% down to 0.8% (see Section 4.6).

As with the gym sequence, we can use the ground plane-induced homography to resample one of the background images into the other's coordinate system. Because both of the images contain the ground plane's line at infinity, the melded images are large. A lightly cropped meld of the background images is shown in Figure 5.18. Note the fairly straight line formed at the base of the building – that this line overlaps inside the melded region and continues straight on outside the melded region indicates that the homography is correct.

The large melded image is further cropped down in Figure 5.19. In that smaller cropped image it appears that there is a line in the grass.

Figure 5.18: A lightly cropped section of the arena background images melded using the final algorithmically-found homography.

This line is an artifact of the melding algorithm, and essentially marks the left-hand edge of the field of view of camera A, as seen by camera B. Also note the location of the sewer (the brown patch in the centre of the grass). It overlaps quite well because the homography is correct and because it is near the centre of the projection.

Figure 5.19: A further cropped section of Figure 5.18.

The same errors in the projected locations that were found in the other sequences are present in this sequence too. Those errors, due to the difference between where the algorithm found the feet and where the human marked the feet, were discussed above as they related to the gym se-

quence. They do not affect meta-target creation, since they are errors of visual interpretation rather than signs of errors in the algorithm.

5.3 Occlusions

Occlusions were simulated at various times in all sequences. This was done by interrupting the algorithm at various times after the first homography had been found, deleting the list of meta-target associations, and then re-starting the algorithm. This simulated a complete loss of track information, since all target history was deleted. Also, in some cases the masks of targets were adjusted to simulate partial occlusions.

In all cases the homography-based multiple-camera tracking system correctly and immediately recovered the correct meta-target associations. This testing indicates that the algorithm is highly resistant to both partial and full target occlusions.

Chapter 6

Conclusions and future work

6.1 Conclusions

The goal of this research, broadly stated, was to solve the consistent labelling problem for pairs of arbitrary-modality static overlapping cameras sharing a common ground plane, with no calibration and automatic learning. The algorithms that were implemented for this thesis have the following characteristics:

- The single-camera tracker works for long RGB video sequences. The background subtraction model allows it to handle objects stopping and becoming part of the background as well as background objects beginning to move as foreground objects.
- To change modalities, the only component that needs to be changed

is the background subtraction function. So long as that function can tell what is different between the current frame and the background model, the whole system will work with non-RGB modalities.

- Field of view lines are found without using any operator input. No calibration is required. The only element required in the scene is the existence of a common ground plane, but neither that plane nor any other part of the scene requires any special calibration pattern.

- The homography-based tracker quickly learns the geometry of the scene, and was shown to create meta-target associations with scaled symmetric transfer distances well below the maximum threshold value used in this thesis.

- The only operator input needed is to decide whether the pair of cameras contains an overlapping field of view.

In addition to the implementation of the full tracking system, this thesis contributed the following elements to the field of computer vision and target tracking:

- A method of finding the pixel location corresponding to the feet of a target was introduced and tested. The method was shown to be slightly better than the bounding box method for normal upright cameras. For tilted cameras the new method was shown to be significantly better, allowing the multiple-camera tracking module to

create additional meta-target associations that would not have been found with the old bounding-box method.

- A method to use trails of markers to calculate a plane-induced homography was introduced. This method reduces the ability of a few badly-segmented targets to ruin the homography, thereby increasing the capabilities of the multiple-camera tracking function. In addition, it enables the multiple-camera tracking function to operate across the whole frame even if only one edge of the field of view is used as an entry and exit point.

- A homography-based multiple camera person tracking algorithm was introduced. The rules and methods used in the operation of the tracker were fully specified. The tracker solves the consistent labelling problem for all targets visible in the scene, including ones that travel through in-scene entrances and exits. The tracker uses a scaled version of the symmetric transfer distance, which is usable when the cameras have differently-sized frames.

The algorithms that were discussed and implemented for this research therefore satisfy the goals stated in Chapter 1.

6.2 Future work

6.2.1 Specific implementation ideas

The algorithms that were implemented for this thesis could be improved in a variety of ways. The segmentation algorithm used in the single-camera tracking algorithm is slow, even when the interpreted nature of MATLAB code is taken into account. Tightly-coded implementations of the segmentation algorithm might be too slow for live video processing. Therefore, another segmentation algorithm should be investigated.

To prove that the system works with other modalities, video should be acquired and processed using a different type of camera. For instance, to use a thermal camera, the two changes required are to re-implement the background-subtraction algorithm and the image segmentation algorithm.

When a target's feet are partially occluded they might not get properly associated into a meta-target, since their "feet" will be detected at an incorrect location. This might also cause meta-target stealing, where a target appears close to another target's knees, but not their feet. It may be possible to use the historical median apparent height of the target to find a putative location for the feet feature, and use that location for meta-target association purposes. This might prevent meta-target stealing problems that could occur when targets are moving in a region where their lower

bodies are sometimes partially occluded, such as a food court or a café.

In its present implementation, the algorithm works discretely in this sequence: acquisition, segmentation, single-camera tracking, then multiple-camera tracking. Ideally this would be a continuous process without interruption. The code, as written, is amenable to being integrated into a continuous process, however, speed would be an issue and the segmentation code is proprietary. In the future, we would like to see the entire algorithm running on live data on a single machine. To do this it is likely that the whole system would have to be re-implemented in a language other than MATLAB. Luckily, nearly all of the code uses fairly generic functionality, not proprietary Mathworks tool-box functions. The only exception is the segmentation algorithm. The most complicated functions outside the segmentation algorithm are a singular value decomposition and a few morphological operations (dilation, opening).

6.2.2 Computer vision

As discussed in Chapter 2, work is being done by other research groups to automatically discover the relationship between cameras. The algorithms in this thesis require that this be specified. It should be possible to integrate the methods developed in this thesis with a network-discovery function. This would improve the in-scene performance of the meta-target creation algorithms, since they would use a better homography-

based tracker.

If a meta-target has been properly segmented, then the head could be considered a corresponding feature point that does not lie on the ground plane. With two or more such corresponding points the epipoles and the fundamental matrix could be calculated, thereby defining the epipolar geometry. If the fundamental matrix is known then the epipolar line upon which the head feature should appear can be calculated. The distance of the target's head to this epipolar line could be incorporated into the scaled symmetric transfer distance metric, where it would therefore be used for meta-target matching. This would enable better matching and selection of close targets.

Additional ideas for systems that could be implemented were discussed in Section 3.4. Those ideas included systems that directly improve the method discussed in this thesis and methods that use the fundamental matrix or other geometric constructs.

Bibliography

[1] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *ACM Computing Surveys (CSUR)*, vol. 38, no. 4, 2006.

[2] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 564–577, May 2003.

[3] S. Sclaroff and J. Isidoro, "Active blobs: region-based, deformable appearance models," *Computer Vision and Image Understanding*, vol. 89, no. 2-3, pp. 197–225, 2003.

[4] A. Koschan, S. Kang, J. Paik, B. Abidi, and M. Abidi, "Color active shape models for tracking non-rigid objects," *Pattern Recognition Letters*, vol. 24, pp. 1751–1765, 2003.

[5] A. Martin and E. Saber, "An improved method for dynamic object tracking using partial shape matching and color image segmentation," January 2008, submitted to IEEE International Conference on Image Processing 2008.

[6] E. Saber, Y. Xu, and A. M. Tekalp, "Partial shape recognition by sub-matrix matching for partial matching guided image labeling," *Pattern Recognition*, vol. 38, pp. 1560–1573, 2005.

[7] L. Garcia, E. Saber, V. Amuso, and R. Bhaskar, "Automatic color image segmentation by dynamic region growth and multimodal merging of color and texture information," in *International Conference on Acoustics, Speech and Signal Processing*, March 2008.

[8] D. Makris, T. Ellis, and J. Black, "Bridging the gaps between cameras," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, July 2004, pp. 205–210.

[9] T. Yang, F. Chen, D. Kimber, and J. Vaughan, "Robust people detection and tracking in a multi-camera indoor visual surveillance system," in *IEEE International Conference on Multimedia and Expo*, July 2007, pp. 675–678.

[10] C. Stauffer and K. Tieu, "Automated multi-camera planar tracking correspondence modeling," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1. IEEE Computer Society, 2003, p. 259.

[11] O. Javed, Z. Rasheed, K. Shafique, and M. Shah, "Tracking across multiple cameras with disjoint views," in *IEEE International Conference on Computer Vision*, vol. 2, October 2003, pp. 952–957.

[12] O. Javed, K. Shafique, and M. Shah, "Appearance modeling for tracking in multiple non-overlapping cameras," in *IEEE International Conference on Image Processing*, vol. 2. IEEE Computer Society, 2005, pp. 26–33.

[13] O. Javed, K. Shafique, Z. Rasheed, and M. Shah, "Modeling inter-camera space-time and appearance relationships for tracking across non-overlapping views," *Computer Vision and Image Understanding*, vol. 109, no. 2, pp. 146–162, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/B6WCX-4N4YMR5-1/1/14477a55fa6ebb45f4713fe74f71be28

[14] J. Orwell, P. Remagnino, and G. Jones, "Multi-camera color tracking," in *IEEE Workshop on Visual Surveillance*. Los Alamitos, CA, USA: IEEE Computer Society, 1999, p. 14.

[15] J. Li, C. S. Chua, and Y. K. Ho, "Color based multiple people tracking," in *International Conference on Control, Automation, Robotics and Vision*, vol. 1, December 2002, pp. 309–314.

[16] F. Devernay, D. Mateus, and M. Guilbert, "Multi-camera scene flow by tracking 3D points and surfels," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 2203–2212, 2006.

[17] C. Madden and M. Piccardi, "Height measurement as a session-based biometric for people matching across disjoint camera views," in *Proceedings of the Conference of Image and Vision Computing New Zealand*, November 2005.

[18] ——, "A framework for track matching across disjoint cameras using robust shape and appearance features," in *IEEE Conference on Advanced Video and Signal Based Surveillance*, September 2007, pp. 188–193.

[19] W. Hu, M. Hu, X. Zhou, T. Tan, J. Lou, and S. Maybank, "Principal axis-based correspondence between multiple cameras for people tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 663–671, April 2006.

[20] D. Thirde, M. Borg, J. Ferryman, J. Aguilera, M. Kampel, and G. Fernandez, "Multi-camera tracking for visual surveillance applications," in *Computer Vision Winter Workshop*, O. Chum and V. Franc, Eds. Czech Pattern Recognition Society, February 2006.

[21] T. Zhao, M. Aggarwal, R. Kumar, and H. Sawhney, "Real-time wide area multi-camera stereo tracking," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1. IEEE Computer Society, 2005, pp. 976–983.

[22] J. Black and T. Ellis, "Multi camera image tracking," *Image and Vision Computing*, vol. 24, no. 11, pp. 1256–1267, 2006.

[23] S. Velipasalar and W. Wolf, "Recovering field of view lines by using projective invariants," in *International Conference on Image Processing*, vol. 5, October 2004, pp. 3069–3072.

[24] S. Khan and M. Shah, "Consistent labeling of tracked objects in multiple cameras with overlapping fields of view," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 10, pp. 1355–1360, 2003.

[25] S. Calderara, A. Prati, R. Vezzani, and R. Cucchiara, "Consistent labeling for multi-camera object tracking," *Image Analysis and Processing*, pp. 1206–1214, 2005.

[26] Z. Yue, S. Zhou, and R. Chellappa, "Robust two-camera tracking using homography," *IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, pp. 1–4, May 2004.

[27] A. Mittal and L. Davis, "Unified multi-camera detection and tracking using region-matching," in *Proceedings of IEEE Workshop on Multi-Object Tracking*. IEEE Computer Society, 2001, pp. 3–10.

[28] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati, "Statistic and knowledge-based moving object detection in traffic scenes," in *IEEE Intelligent Transportation Systems Conference Proceedings*, October 2000, pp. 27–32.

[29] R. Cucchiara, C. Grana, M. Piccardi, A. Prati, and S. Sirotti, "Improving shadow suppression in moving object detection with hsv color information," in *IEEE Intelligent Transportation Systems Conference Proceedings*, 2001, pp. 334–339.

[30] R. Cucchiara, M. Piccardi, and A. Prati, "Detecting moving object, ghosts, and shadows in video streams," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 10, pp. 1337–1342, October 2003.

[31] R. H. Bartels, J. C. Beatty, and B. R. Barsky, *An introduction to splines for use in Computer Graphics and Geometric Modeling*, 2nd ed. Morgan Kaufmann Publishers Inc., 1987.

[32] A. Criminisi, I. Reid, and A. Zisserman, "Single view metrology," in *Fifth International Conference on Computer Vision*, vol. 1. IEEE Computer Society, 1999, p. 434.

[33] J. Grimm and W. Grimm, *The Complete Grimm's Fairy Tales*, M. Hunt, Ed. Pantheon Books, 1944.

[34] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[35] R. Hartley, "In defence of the 8-point algorithm," in *International Conference on Computer Vision*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 1995, p. 1064.

[36] (2008, May) Performance evaluation of tracking and surveillance. [Online]. Available: http://www.cvg.cs.rdg.ac.uk/cgi-bin/PETSMETRICS/page.cgi?dataset

[37] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[38] P. D. Kovesi, "MATLAB and Octave functions for computer vision and image processing," School of Computer Science & Software Engineering, The University of Western Australia, February 2008, available from: <http://www.csse.uwa.edu.au/~pk/research/matlabfns/>.